University of California
Santa Barbara

Exploiting Intrinsic Clustering Structure in Discrete-Valued Data Sets for Efficient Knowledge Discovery in the Presence of Missing Data

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Veronika Strnadová-Neeley

Committee in charge:

Professor John R. Gilbert, Chair
Professor Linda Petzold
Professor Xifeng Yan

September 2018

The Dissertation of Veronika Strnadová-Neeley is approved.

Professor Linda Petzold

Professor Xifeng Yan

Professor John R. Gilbert, Committee Chair

May 2018

Exploiting Intrinsic Clustering Structure in Discrete-Valued Data Sets for Efficient Knowledge Discovery in the Presence of Missing Data

by

Veronika Strnadová-Neeley

To my parents, who gave me the confidence to shoot for the moon,

remind me to respect the stars,

and will never hesitate to smack me back down to earth.

Acknowledgements

It is difficult to overstate the amount of support I have received over the course of my graduate school years. I will do my best to give due credit to the many people who have enabled me to complete this journey.

I thank my adviser, John R. Gilbert, for enormous support and guidance. John is at the same time a successful and brilliant academic and one of the most humble, generous and kindest people I have ever met. CSC Lab provided a great environment for me to grow as a researcher. Lawrence Berkeley National Laboratory was a second academic home for me for many years, and I thank the lab for providing me with the resources and support to conduct my research. I thank Aydın Buluç, Jarrod Chapman, Leonid Oliker, Joseph E. Gonzalez, Stefanie Jegelka, and Daniel S. Rokhsar for their constant encouragement and mentorship during my years at Berkeley Lab.

I've shared many valuable research discussions with Kevin Deweese and I am thankful for our strong friendship. I thank Maggie Wang for her encouragement and her competitive and uplifting spirit. Saiph Savage was a role model and a friend who motivated me to match her incredible work ethic. Roya Ensafi has offered invaluable support and advice throughout our long friendship. Emily Finley was a source of knowledge and strong support in all matters related to graduate school.

I have had tremendous help and support from my sisters and parents. I am lucky to have a family that constantly offers love and support. My husband, Jaime Neeley, has been a source of unwavering support, and I will always be grateful for his love, patience, and optimism throughout our years before and during graduate school. His commitment to making our marriage an equal partnership is rare and sincere. I thank Sierra, my toddler, for her loving heart and easygoing attitude, and for the many joys of parenting that put the academic life in perspective.

Curriculum Vitæ

Veronika Strnadová-Neeley

Education

2018	Ph.D. in Computer Science, University of California, Santa Barbara.
2016	M.S. in Computer Science, University of California, Santa Barbara.
2011	B.S. in Applied Mathematics, *summa cum laude* University of New Mexico

Publications

Strnadová-Neeley, V., Gilbert, J. R. (2017, September) Efficient Clustering for Large-Scale, Sparse, Discrete Data with Low Dimensional Intrinsic Structure. (Regular Presentation) In *CEUR Workshop Proceedings (VLDB17 PhD Workshop)*, Vol. 1882.

Strnadová-Neeley, V., Buluç, A., Gilbert, J. R., Oliker, L., & Ouyang, W. (2016). LiRa: A New Likelihood-Based Similarity Score For Collaborative Filtering. *arXiv preprint arXiv:1608.08646.* Presented in Workshop on Large Scale Recommendation Systems (LSRS16)

Strnadová-Neeley, V., Buluç, A., Chapman, J., Gilbert, J. R., Gonzalez, J., & Oliker, L. (2015, September). Efficient data reduction for large-scale genetic mapping. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics* (pp. 126-135). ACM.

Chapman, J. A., Mascher, M., Buluç, A., Barry, K., Georganas, E., Session, A., **Strnadová, V.**, Jenkins, J., Sehgal, S., Oliker, L. & Schmutz, J. (2015). A whole-genome shotgun approach for assembling and anchoring the hexaploid bread wheat genome. Genome biology, 16(1), 26.

Strnadová, V., Buluç, A., Chapman, J., Gilbert, J. R., Gonzalez, J., Jegelka, S., ... & Oliker, L. (2014, November). Efficient and accurate clustering for large-scale genetic mapping. In *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on* (pp. 3-10). IEEE.

Strnadová, V., Jurgens, D., & Lu, T. C. (2013, March). Characterizing Online Discussions in Microblogs Using Network Analysis. In *AAAI Spring Symposium: Analyzing Microtext*.

Awards

2018	UCSB CS Department Outstanding Graduate Student Award
2017	UCSB CS Department Grace Hopper Conference Scholarship
2015	Selected participant for the M.I.T. EECS Rising Stars Workshop
2014	IEEE Student Travel Award, IEEE International Conference on Bioinformatics and Biomedicine
2013	SIAM Student Travel Award, Parallel Processing for Scientific Computing (PP14)
2012, 2013	Google Anita Borg Memorial Scholarshio Finalist
2012	One of Four Best Reviewer Awards for the UCSB CS-GSA Student Conference
2011	NCAA 1A FAR Academic Excellence Award
2006-2010	Presidental Scholarship Recipient
2006-2010	Mountain West Conference Scholar Athlete

Research and Teaching Experience

2011 - 2018 **Graduate Research Assistant in Dr. John R. Gilbert's CSC Lab at UC Santa Barbara**

- Collaborated with Lawrence Berkeley National Lab researchers on data analysis for genetic mapping and a new similarity score for recommender systems

2018 **Teaching Associate at UCSB for CS140: Scientific Parallel Computing**

- Lead instruction for the CS140 course at UCSB. Designed homework and tests, managed two teaching assistants.

2013 - 2016 **Graduate Research Assistant at Lawrence Berkeley National Laboratory, managed by Leonid Oliker**

- Worked on clustering algorithms for genetic linkage group discovery and data reduction for genetic mapping and a new similarity score for recommender systems

2015 **Instructor of Record for CS8: Introducton to Computer Science at UC Santa Barbara**

- Designed and taught an introductory comouter science course to aooroximately 80 students, managed three teaching assistants

2014 **Teaching Assistant for CS240A: Applicatons of Parallel Programming at UC Santa Barbara**

- Taught 3 individual lectures while Dr. Gilbert was away on travel, held a C tutorial for beginners, held office hours every

week, graded all the homework for a class of 40, helped grade final project proposal, progress reports, presentations and demos, assisted our supercomputing consultant in updating the Triton supercomputer web page

2012 **Intern with the Informaton Systems and Sciences Lab at HRL**

- Work resulted in publication at the 2013 AAAI Spring Symposium on Analyzing Microtext

2011 **Teaching Assistant for CS111: Introduction to Scientific Computing at UC Santa Barbara**

- Taught one 1-hour section per week, graded half of all assignments & midterms, held office hours weekly, attended weekly seminars on Teaching Computer Science

Service and Volunteer Work

2017	Program Committee Member for LSRS17: Large-Scale Recommendation Systems Workshop
2016	Panel Chair, UCSB CS Summit
2015 - 2016	President of the UCSB Computer Science Graduate Student Representatives
2015 - 2016	Diversity Committee Member, Computer Science Department at UCSB
2015	Reviewer for ACM IUI 2016
2014 - 2015	Undergraduate Affairs Committee Member, Computer Science Department at UCSB
2014 - 2015	G.U.I.D.E.S. Mentor at UCSB
2014	Reviewer for IEEE Transactions on Knowledge and Data Engineering
2013 - 2018	Associate, IGERT Network Science Program at UCSB
2013 - 2014	Mentorship Committee Member, Women in Science and Engineering at UCSB
2012 - 2013	Co-President, Women in Computer Science at UCSB
2011 - 2012	Officer, Women in Computer Science at UCSB

Professional Affiliations

SIAM	Member since 2013
IEEE	Member since 2014
ACM	Member since 2015

Abstract

Exploiting Intrinsic Clustering Structure in Discrete-Valued Data Sets for Efficient Knowledge Discovery in the Presence of Missing Data

by

Veronika Strnadová-Neeley

Scalable algorithm design has become central in the era of large-scale data analysis. The vast amounts of data pouring in from a diverse set of application domains, such as bioinformatics, recommender systems, sensor systems, and social networks, cannot be analyzed efficiently using many data mining and statistical tools that were designed for a small scale setting. It is an ongoing challenge to the data mining, machine learning, and statistics communities to design new methods for efficient data analysis. Confounding this challenge is the noisy and incomplete nature of real-world data sets. Research scientists as well as practitioners in industry need to find meaningful patterns in data with missing value rates often as high as 99%, in addition to errors in the data that can obstruct accurate analyses.

My contribution to this line of research is the design of new algorithms for scalable clustering, data reduction, and similarity evaluation by exploiting inherent clustering structure in the input data to overcome the challenges of significant amounts of missing entries. I demonstrate that, by focusing on underlying clustering properties of the data, we can improve the efficiency of several data analysis methods on sparse, discrete-valued data sets. I will highlight new methods that I have developed with my collaborators for three diverse knowledge discovery tasks: (1) clustering genetic markers into linkage groups, (2) reducing large-scale genetic data to a much smaller, more accurate representative data set, and (3) computing similarity between users in recommender systems In

each case, I will point out how the underlying clustering structure can be used to design more efficient algorithms, even when high missing value rates are present.

Contents

Chapter 1

Introduction

This thesis focuses on exploiting underlying cluster structures in large, discrete-valued data sets, in order to enable efficient clustering or similarity comparison in the presence of large amounts of missing values. These clustering structures are explored in detail and demonstrated to be crucial to the performance of clustering, data reduction, and k-nearest neighbor algorithms that are applied to achieve domain-specific goals. To demonstrate the merits of this approach, I will focus heavily on two application domains where input data is often discrete-valued, and missing data is abundant: genetic mapping and recommender systems. In this chapter, I will motivate the necessity for a closer examination of underlying clustering structure in large-scale, discrete-valued data. I will also provide some background describing the state of affairs of specific data mining tasks in recommender systems and genetic mapping, and place my work into context.

1.1 The missing data predicament in understanding large-scale, discrete-valued data

The necessity for efficient algorithms in large-scale data analysis has become clear in the past few years, as unprecedented scales of information have become available in a

variety of domains, from bioinformatics to social networks to signal processing. In many cases, it is no longer sufficient to use even quadratic-time algorithms for such data, and much of recent computer science research has focused on developing efficient methods to analyze vast amounts of information.

In this work, I focus on developing efficient knowledge discovery methods for large-scale, discrete valued data, with many missing values. Such data is ubiquitous in the domains of genetic mapping and recommender systems, which form the application foci of this thesis. Input data in these domains is often represented as a discrete-valued m by n matrix, as illustrated in Figure 1.1, with missing entries represented by dashes.

As I detail in sections 1.2 and 1.4, knowledge extraction from this type of data often involves finding highly similar pairs or groups of vectors (either rows or columns of the input matrix) in the data set. Thus, my efforts have in large part been focused on improving or devising new, efficient *clustering* methods for large-scale, discrete-valued data with many missing values.

Clustering is a form of unsupervised learning that is invaluable in exploratory data analysis [1, 2]. The goal of a clustering algorithm is to find groups of similar items in a data set. Applications of clustering abound – social and political scientists, biologists, and chemists have used clustering algorithms to identify communities in social networks [3], find linkage groups in genomes [4, 5], and group molecules by structural similarity [6, 7]. Currently, challenges to designing good clustering algorithms arise due to the real-world messiness of input data in many applications: the data is often of large scale, high dimensionality, contains many missing or unknown values, and/or is cluttered with noise.

Two noteworthy factors contribute to the difficulty of finding clusters in data sets with incomplete and discrete-valued data. First, any algorithm that attempts to assess similarity between two vectors necessitates the selection of an appropriate similarity

Input Data Matrix

x_1	1	2	-	-	4
x_2	3	4	1	2	1
x_3	1	2	-	-	-
x_4	3	-	1	-	2
x_5	4	-	2	2	-
x_6	2	2	3	5	-
x_7	-	-	-	2	5
x_8	3	-	2	1	2
x_9	2	1	3	-	3
x_{10}	1	2	2	5	-
x_{11}	3	-	4	-	4
x_{12}	3	4	-	4	-
x_{13}	5	-	5	1	2

Figure 1.1: Discrete-valued input data matrix of size 13 by 5, with many missing values. Matrix entries are constrained to a discrete set of values, and many entries, represented by dashes here, are unknown. Data sets of this type are difficult to reason about, due to the constraints on matrix entries and the incomplete nature of the matrix.

score or measure. The choice of similarity score has been shown to have a large influence on the quality and accuracy of various knowledge extraction tasks such as clustering [8] and identifying near neighbors [9, 10, 11]. Missing data exacerbates this issue, as it is often unclear how to treat missing values when evaluating similarity between two vectors. Indeed, when missing entries are common, it has been necessary to derive domain-specific scores that provide a reliable estimate of similarity between a pair of data points [5, 12, 13, 14]. Standard methods for addressing the missing data issue, such as multiple imputation or complete-case analysis can be costly to implement and have been shown to perform poorly in several domains where missing data is abundant [15]. Second, analyzing data that is discrete in nature often limits the data analysis techniques that can be applied to understand such data. For the clustering task in particular, these limits pertain to the choice of similarity score as well as clustering algorithm [16].

An illustrative example of the challenges in clustering discrete-valued data. is the dated but still widely-used k-means algorithm [17]. The k-means algorithm cannot be applied directly to discrete-valued input such as categorical or ordinal data, because it does not make sense to take averages or "means" in this case. In addition, the dissimilarity between discrete-valued data points may not obey the triangle inequality. Thus, to use a similar centroid-based clustering approach, one must instead choose between alternatives such as the k-mediods [18], k-modes [19] or k-medians [20] methods. Although these are useful alternatives, they are far less understood and less thoroughly tested than the k-means algorithm on a variety of problems in many domains.

Although the combination of discrete values and missing data hinders our ability to analyze large-scale data sets of this type, I will show that a focus on underlying structure leads to insights that enable efficient, accurate algorithms for data analysis. The applicability and generalizability of this approach is demonstrated in two important yet fundamentally different domains. I will present algorithms for clustering, data reduction,

and similarity computation in k-nearest neighbor algorithms in the context of genetic mapping and recommender systems. By exploiting inherent low-dimensional structure in the input data, the methods proposed here overcome the challenges of significant amounts of missing entries and are even robust to some noise. In the genetic mapping domain, I will show that exploiting a locally linear structure leads to an efficient clustering algorithm that produces accurate results even in cases with up to 65% missing data and errors in the input. This work is presented in Chapter 2. I will also present a data reduction method for large-scale genetic map data in Chapter 3, at the center of which is a novel notion of low-dimensional underlying structure that we call *fundamental resolution*. In Chapter 4, I will discuss the concept of fundamental resolution more generally in the discrete-valued data setting, and show how this concept can be applied in the domain of recommender systems with the development of a novel similarity score for collaborative filtering. Chapter 5 concludes this thesis and presents several promising future work directions.

1.2 General work on scalable algorithms with a focus on underlying structure

The methods presented in this thesis are aimed at improving the scalability of knowledge discovery in large-scale, sparse, discrete-valued data, and largely center around the task of finding clusters in a data set. Sections 2.3.4, 3.5 and 4.4 provide a detailed overview of the most highly related work to each of clustering, data reduction, and similarity score computation methods of my thesis. Here, I will overview some widely used algorithms that share the goal of finding clusters efficiently in large-scale data, with an emphasis on underlying structure in the data albeit in more general settings than the

genetic mapping and recommender system domains.

Many successes in the effort to design scalable algorithms have focused on leveraging an inherent structure in the data, and its structure may be best expressed in various ways. The data may be best described as lying in an inherently low-dimensional Euclidean space, along a low-dimensional manifold, or it may have certain self-repeating, or *fractal* properties. All these structural properties have been explored to some degree in order to design more efficient clustering algorithms.[21, 22, 23, 24, 25]

Perhaps the most popular clustering algorithm is still the k-means algorithm, and the many variations thereof. In the k-means problem, the structural assumptions about underlying clusters is that (1) there are a fixed number of clusters k that do not overlap and (2) each cluster has a (hyper-)spherical shape around the cluster *mean* or *centroid*, that is the the mean of all (possibly high-dimensional) points within the same cluster. Given an input parameter k and n-dimensional data $D = \{x_1, ..., x_m\}, x_i \in R^n$, the k-means algorithm finds k means $\mu_1, ..., \mu_k \in R^n$, along with assignments of each x_i to one of the means μ_j, such that the distances between points x_i and their assigned means are small. More formally, the k-means algorithm finds a local minimum of the objective function that is the sum of squared distances between all x_i and their assigned means:

$$k\text{-means objective} = \sum_{j=1}^{k} \sum_{i=1}^{m} \delta_{ij} (||x_i - \mu_j||_2)^2$$

where δ_i is an indicator function with value 1 if x_i is assigned to mean μ_j, and 0 otherwise. Minimizing this function is NP-hard in general, but the local minimum found by the k-means algorithm has been found to produce useful and meaningful results in practice. Starting with an initial assignment of the m points to k clusters, the algorithm then iterates two steps until convergence: (1) re-compute the mean of each cluster based on the updated cluster assignments, and (2) assign each point to the cluster whose mean

is closest in Euclidean distance. The number of iterations to find a clustering is not bounded in theory, but often terminates quickly in practice.

There are a myriad of improvements and extensions to the k-means algorithm, including variants that aim to improve the initialization schemes of the k means, such as the k-means++ algorithm [26] that can lead to a provably better clustering in terms of the objective function. However, missing data has proven challenging to address, even in the k-means setting. As Chi and Chi note in the description of their recently developed k-POD [15] algorithm: "Mainstream approaches to clustering missing data reduce the missing data problem to a complete data formulation through either deletion or imputation but these solutions may incur significant costs." Their k-POD algorithm is shown to outperform modern imputation techniques, in terms of both clustering quality and computation time, when solving the k-means problem on a few data sets tested with a wide range of missing values. The k-POD algorithm is a testament to the challenges in clustering data with missing values, as it has taken over fifty years to extend perhaps the most widely known and studied clustering algorithm to the missing data setting. Though it was shown to perform fairly well in comparison to data imputation and deletion techniques, it has yet to be evaluated in the large scale setting, and the authors admit that it suffers from the same drawbacks as the original k-means algorithm. For example, the k-POD algorithm is expected to "struggle when the scatter within-clusters varies drastically from cluster to cluster," again highlighting the need for a focus on underlying structure in data mining algorithm design.

Besides the many variants to solving the k-means problem, density-based approaches to clustering have also been popular and applicable to large-scale data. In these methods, the underlying structural assumption is that clusters are regions of high density and proximity of input points. Ester et al. introduced the still widely used DBSCAN[23] algorithm in 1996, with the goal of quickly finding arbitrarily-shaped clusters in large-

scale data. DBSCAN has proven invaluable to the data mining community, as evidenced by its receiving the test of time award [1] in 2014, and is a common algorithm of choice for clustering large-scale data.

DBSCAN starts with an initial seed, then grows clusters by adding points to the same cluster as the seed as long as the points are within a fixed radius ϵ of the seed. If the added points are in a region of high enough *density*, then the same process is repeated for these points. Density is defined as the number of points within the fixed radius ϵ of the seed, and both the density threshold and the radius are input parameters of the algorithm. Once there are no more points whose neighborhoods need to be explored, a new seed is selected from the input data and a new cluster is found in the same manner. In 2013, Campello et al. introduced a hierarchical version of DBSCAN which addresses some limitations of the original algorithm, such as its inability to find clusters of varying densities, as well as its propensity to misclassify noise [27].

Some controversy has surrounded assumptions and claims about the running time of DBSCAN, however. In 2015, Gan et al. and disputed the common assumption that the complexity of DBSCAN is just $O(n \log n)$ in general, showing that the algorithm requires $O(n^{4/3})$ time if the input points have dimensionality greater than or equal to 3 and Euclidean distance is used[28]. Others have shown that DBSCAN has a worst-case complexity of $O(n^2 d)$ in the general case, with n d-dimensional input points and an arbitrary distance function. The authors of the original DBSCAN paper have responded to these challenges, acknowledging that DBSCAN indeed requires $O(n^2 d)$ time in the general case, and $O(n^{4/3})$ time in the Euclidean setting with dimensionality ≥ 3. Despite these bounds, the authors go on to discuss and empirically show why DBSCAN is nonetheless a good choice of clustering algorithm on many large, real-world data sets [29]. The popularity and controversy of DBSCAN point to the difficulty of efficiently finding

[1] `http://www.kdd.org/News/view/2014-sigkdd-test-of-time-award`

clusters in large-scale data, even with relatively simple assumptions on the underlying cluster structure. As DBSCAN relies on searching ϵ-neighborhood around input points, the handling of missing data is delegated to the indexing structure used for finding near neighbors in practice.

Standard matrix decomposition-based methods such as spectral clustering [8, 21], and principal component analysis combined with a standard clustering algorithm such as k-means [30], rely on a singular value decomposition (SVD) of the similarity or covariance matrix of (centered or 0-mean, in the case of PCA) input data. The SVD is used to project the data onto a lower dimensional, linear subspace. When used for clustering, it is assumed that clusters will be well-separated and easily found after projecting the data onto this low-dimensional space.

As discussed in section 1.1, constructing a similarity matrix for data with many missing values is difficult and necessitates the choice of an appropriate similarity score that accounts for missing data. Likewise, the SVD of a matrix is undefined when any of the matrix entries are missing.

The CUR decomposition [31] is another flavor of matrix-decomposition, which differs from SVD in that the decomposition is made up of actual rows and columns of the original data, not of eigenvectors, which are linear combinations of the same rows and columns and often not interpretable in practice. The CUR has been used in the same way as the SVD for clustering – data is projected onto a lower-dimensional space obtained by taking the most significant rows or columns of the CUR decomposition, and it is assumed that clusters are then easily found in this lower-dimensional space. It remains a challenge to extend CUR decompositions to the case when much of the data is missing.

Several techniques have been developed to preserve more complicated, nonlinear structure of clusters in dimensionality reduction. Examples include kernel PCA [32], locally linear embedding [33], Laplacian eigenmaps [22]. In general, it is still an active research

problem to find efficient dimensionality reduction methods that preserve nonlinear structure in the missing data realm. [34]

An interesting and less well-known underlying structure of large scale data that has been explored in the context of clustering is fractal dimension [35, 24, 36, 25] The fractal dimension, which characterizes the extent to which a data set is self-similar, or self-repeating, has received some attention in the past due to its applicability to detecting various low-dimensional structures in high-dimensional data [35, 24]. Yu et al. [25] have shown that a data set with small fractal dimension implies low search time in massive biological data sets, assuming a roughly uniform density of data points exists over the entire data set. Hoecker et al. have developed a fractal-based similarity score that has been tested on astronomical data, which works well to identify whether a point belongs in a given cluster after an initial clustering [36]. However, very little work has been done on general clustering methods for data with low fractal dimension, or for utilizing fractal dimension in order to cluster large-scale, sparse data.

To summarize, the landscape of clustering algorithms too vast and diverse to survey here, but some of the most popular clustering approaches highlight the benefit of carefully considering underlying clustering structure in large-scale data to efficient algorithm design. However, extending these methods, or inventing new techniques, to address the missing data problem in large-scale data remains a challenge. In sections 1.3 and 1.4, I will discuss the problems stemming from missing data in the genetic mapping and recommender systems domains and I will introduce the ways I have leveraged underlying clustering structure to help alleviate these problems.

1.3 Analyzing large-scale genetic map data

A significant portion of my work has been aimed at improving computational genetic mapping tools by designing new algorithms for large-scale data in this domain. Here, I will give a brief introduction to the genetic mapping problem, explain why new, scalable algorithms for data analysis are required, and introduce the contributions I've made toward this goal.

Genetic maps are essential tools for analyzing DNA sequence data, not only providing a blueprint of the genome but also unlocking linkage patterns between *genetic markers*, chromosomal regions with two or more sequence variants in a population.

Genetic maps are essential for organizing DNA sequence information along chromosomes, and they enable diverse applications of genetics to problems in health, agriculture, and the study of biodiversity. Early genetic maps were constructed using only a few hundred genetic markers, and with such limited data, their construction was accordingly computationally inexpensive. With the advent of inexpensive high-throughput "next generation" sequencing [37], however, it is becoming a simple matter to generate data corresponding to millions of genetic markers across a genome. This flood of data rules out many standard, slow genetic mapping algorithms and poses a new challenge: to produce accurate high-density genetic maps in a computationally efficient manner.

In the past few years, next generation sequencing technologies [37] have led to tremendous growth in the amount of genetic markers available to geneticists. However, our ability to parse, analyze, and extract knowledge from this data has not kept pace with the power to generate it. Despite advances in de-novo genome assembly, genetic maps are still critical to many non-trivial assemblies. Genetic maps are commonly used to complete the assemblies of complex and repetitive genomes, by anchoring many disconnected scaffolds to chromosomes.

A genetic map is a linear ordering of genetic markers that is consistent with observed patterns of inheritance in a population. An essential concept is the *linkage group* which collects together markers that are found on a single chromosome. Genetic maps are therefore organized into multiple linkage groups, with the number of groups equal to the number of chromosomes in the species. Within a linkage group, there is a natural measure of proximity (or genetic linkage, or recombination distance), which arises from the linear structure of chromosomes and the mechanics of their transmission from generation to generation.

Genetic maps can be constructed without detailed knowledge of the contiguous sequence of a genome, as was done for many years in the absence of methods for large-scale genome sequencing. With the advent of high throughput genome sequencing technologies an important new application for genetic mapping has arisen. Genome sequencing and assembly approaches often allow the reconstruction of relatively short genomic stretches spanning tens or hundreds of kilobases (i.e., enough to include one or a few genes). In this way the sequences of most if not all genes of can be recovered through sequencing, but in relatively short unlinked pieces whose chromosomal positions are not known. By integrating large-scale genetic mapping with genome sequencing and assembly, we can produce complete chromosome sequences. This problem is especially common in plant genome studies, where the repetitiveness of plant genomes limits the size of sequence assemblies.

Given a pair of markers in the same linkage group, we can estimate their proximity on the chromosome by comparing their sequence across a *mapping population* of related individuals. This estimate is made based on the LOD score, a logarithm of odds that two markers are genetically linked, based on the similarities and differences across each individual's genotype. The fundamental problem of genetic map construction is to take as input the sequences of n related individuals at m genetic markers, with low genotyping

errors and often missing data (unknown genotypes of particular markers for particular individuals), and to organize these markers into linear chains that represent the structure of chromosomes.

Genetic map data can thus be represented as an m by n matrix as in Figure 1.1, where rows represent genetic markers, and columns represent individuals from a mapping population. The first step of genetic mapping involves clustering markers into linkage groups. This is traditionally performed by various standard clustering algorithms applied to a similarity graph of the markers. The similarity score between two markers can be a simple attribute comparison or a computationally intensive procedure, such as estimating the recombination rate of two genetic markers via nonlinear regression [38, 39]. On large datasets, computing the $O(m^2)$ pairwise similarities between all m markers quickly becomes prohibitive. In addition, the abundance of missing entries in genome sequencing data makes it challenging to translate the LOD score into a distance function that respects the triangle inequality, a requirement imposed on the data by many popular fast clustering algorithms.

After linkage group discovery, the next phase of genetic mapping is the *ordering* of genetic markers within each linkage group. The ordering stage of the genetic mapping pipeline can be solved efficiently using heuristics such as a minimum-spanning tree finding algorithm [39], but is a computational challenge in general, and has connections to the Traveling Salesman Problem. Wu et al. [39], showed that "when the data quality is high, the optimal order [of genetic markers] can be identified very quickly" using their minimum-spanning tree approach. However, with large missing data rates and levels of noise that are typical in high-throughput sequencing, the problem becomes intractable.

I have made two major contributions to knowledge discovery in genetic mapping:

1 A novel clustering algorithm for the linkage-group discovery phase of the genetic

mapping pipeline, that produces high-quality clustering results even with high missing data rates and higher-than-expected levels of noise in the input data. This method is shown empirically to outperform state-of-the art, domain-specific clustering methods for genetic mapping, as well as a fast, general-purpose spectral clustering algorithm. The algorithm is called BubbleCluster and is described in Chapter 2.

2 A new data reduction method, which reduces the large-scale, noisy, and incomplete genetic map data set to a smaller, more complete and accurate set of representative points that better capture the underlying structure of the genetic map. This method is described in Chapter 3.

1.4 K-nearest neighbor collaborative filtering in recommender systems

In the last portion of my thesis, I have expanded on some of the concepts that proved essential to efficient algorithm design in the genetic mapping domain to the domain of recommender systems. Though these domains have few commonalities in terms of scientific or industrial objectives, but they share the data characteristics that are examined in section 1.1 and visualized in Figure 1.1 – data are often discrete-valued, large-scale, of high dimensionality, and fraught with missing values. These characteristics impede the construction of accurate similarity comparisons between input points, which are an essential component of many algorithms for knowledge discovery tasks. In this section, I will give a brief overview of the task of collaborative filtering in the recommender systems domain, I will describe the k-nearest-neighbor approach that has been widely adopted to complete this task in practice, and I will introduce how my contribution of a novel

similarity score for recommender system data improves the accuracy of the k-nearest neighbor method.

Recommender systems arose as a way to provide personalized recommendations, in a setting where the number of available items, products, or options to a user is too large to sift through manually. Collaborative filtering has proven to be an effective approach for recommendation, relying on the similarity of users or items in a system to predict future user preferences. The premise underlying user-based collaborative filtering is that similar users tend to rate items similarly. Therefore, to predict how a user u will rate an item i, we should look at the ratings given to i by users similar to u. In item-based collaborative filtering, the assumption is that similar items tend to be rated similarly by the users. In this case, the rating prediction is based on the ratings given by user u to items similar to i. A central component of any collaborative filtering algorithm is the choice of similarity score that is used to evaluate user-user or item-item similarity.

Despite many improvements and successes of modern model-based collaborative filtering, the k-Nearest-Neighbor (kNN) method remains a popular and widely used approach, in large part due to its simplicity and scalability [40]. To perform user-based collaborative filtering, the kNN method predicts the rating p_{ui} for an item i by a user u by selecting the k most similar users to u who have rated item i. The prediction p_{ui} is then computed by taking a (often weighted) average of the ratings given to item i by these k similar users. Similarly, in item-based kNN, the ratings of the k items most similar to item i and rated by user u are used to compute p_{ui}. Many variants of kNN have been proposed and investigated in order to provide guidelines for optimal parameter settings and implementation choices. The choice of similarity score has consistently been shown to be highly influential on rating prediction accuracy [40, 41, 12, 42].

My contribution to this domain is the design of a new similarity score, that we call LiRa, which avoids some of the common pitfalls of standard similarity scores when applied

to data with many missing values. LiRa is based on the concept that an underlying clustering structure exists among the users represented by recommender system data, and exploits this structure in order to assess user-user similarity in a user-based approach to collaborative filtering. LiRa is presented in Chapter 4. We show that LiRa is more effective in finding the best users to include in a k-nearest neighbor rating prediction scheme in comparison to other, standard scores in recommender systems as well as to a new similarity score designed for the extreme missing data regime.

1.5 Summary of the objectives of this thesis

I have described the domain-specific problems that I will address through a focus on underlying clustering structure: clustering genetic markers into linkage groups, reducing large-scale genetic data to a much more complete and accurate representative set of data vectors, and evaluating similarity between users in order to perform k-nearest neighbor-based rating prediction for a recommender system. The approach taken here begins with a detailed examination of the underlying structure in a data set, before investigating the usefulness of several methods for knowledge discovery. A thorough understanding of the structure in certain large-scale data leads to insights that can lead to more efficient algorithms for the knowledge discovery task. In the following, I will argue that my approach can provide simple, elegant solutions to data mining problems where large amounts of missing values hinder the efficiency or accuracy of general methods.

1.6 Permissions and Attributions

1. The content of chapter 2 is the result of a collaboration with Aydın Buluç, Jarrod Chapman, John R. Gilbert, Joseph E. Gonzalez, Stefanie Jegelka, Leonid Oliker and

Daniel Rokhsar, and has previously appeared in the IEEE International Conference on Bioinformatics and Biomedicine (BIBM14) [43]. It is reproduced here with the permission of IEEE.

2. The content of chapter 3 is the result of a collaboration with Aydın Buluç, Jarrod Chapman, John R. Gilbert, Joseph E. Gonzalez, Stefanie Jegelka, Leonid Oliker and Daniel Rokhsar, and has previously appeared in the 6th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB15) [44]. It is reproduced here with the permission of ACM.

3. Much of the content of chapter 4 is the result of a collaboration with Aydın Buluç, John R. Gilbert, Weimin Ouyang, and Leonid Oliker and was previously presented at the Large-Scale Recommender Systems Workshop (LSRS16).

Chapter 2

Scalable Clustering for Genetic Mapping

High-throughput "next generation" genome sequencing technologies are producing a flood of inexpensive genetic information that is invaluable to genomics research. Sequences of millions of genetic markers are being produced, providing genomics researchers with the opportunity to construct high-resolution genetic maps for many complicated genomes. However, the current generation of genetic mapping tools were designed for the small data setting, and are now limited by the prohibitively slow clustering algorithms they employ in the genetic marker-clustering stage. In this work, we present a new approach to genetic mapping based on a fast clustering algorithm that exploits the geometry of the data. Our theoretical and empirical analysis shows that the algorithm can correctly recover linkage groups. Using synthetic and real-world data, including the grand-challenge wheat genome, we demonstrate that our approach can quickly process orders of magnitude more genetic markers than existing tools while retaining and in some cases even improving the quality of genetic marker clusters.

2.1 Motivation and Background

To the best of our knowledge, no faster clustering method than single linkage has been successfully applied to the linkage group finding phase of genetic mapping, resulting in a troublesome gap between the amount of sequence data available for analysis and the amount that can be efficiently processed by current mapping tools. Our initial efforts at benchmarking the popular genetic mapping tools JoinMap and MSTMap revealed that the clustering stage is indeed a severe bottleneck to the genetic mapping process.

Here we present a fast clustering algorithm that circumvents the computation of all similarities by exploiting prior knowledge about the specific structure of the marker data: linkage groups (i.e., chromosomes) have an intrinsically linear substructure that remains reflected in the similarity measure. After sorting, the algorithm creates a specific sketch that respects both the geometry and quality of the data. We show correctness of the algorithm under mild assumptions, and our empirical evaluation on synthetic and real-world data demonstrates its scalability and accuracy in practice.

2.2 Problem Definition

Computational tools for genetic mapping follow three phases: (1) grouping markers into linkage groups (typically chromosomes), (2) ordering markers within chromosomes and (3) map distance estimation (Fig. 1(a)). Current software tools, such as the popular MSTMap [39] and JoinMap [45], typically fail to scale beyond tens of thousands of markers, especially when there is a high missing data rate. Our initial benchmarks revealed that a severe bottleneck is the pairwise similarity calculation step in the linkage group construction phase, and we therefore focus on this bottleneck here.

We attempt to solve the following problem: given m markers out of a population of

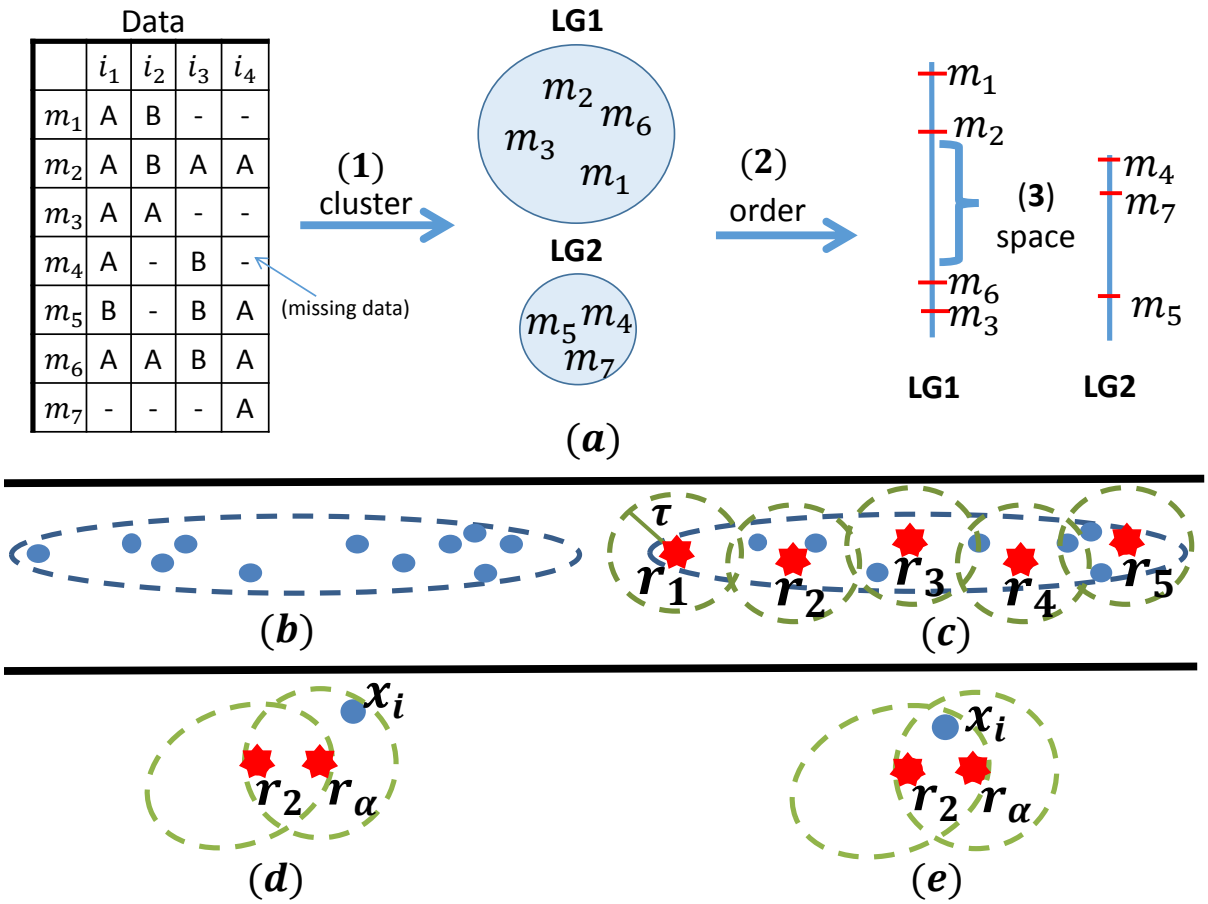

Figure 2.1: **(a)** Genetic map construction pipeline: The markers are clustered into linkage groups (**LG**'s), ordered within each linkage group, and finally spaced according to their genetic distance. **(b)** The linear structure of markers within a linkage group; **(c)** representative points r_i are shown as red stars; **(d)**, **(e)** difference between a point x_i that is added as a new boundary point **(d)** and one that is not **(e)** based on $LOD(r_\alpha, r_2)$, $LOD(r_\alpha, x_i)$ & $LOD(x_i, r_2)$.

n individuals, with a low genotyping error rate and a known missing data rate μ, cluster the markers into a (possibly unknown) number k of clusters. Each cluster represents one linkage group from the species whose sequence data is obtained from the mapping population. Formally, we are given m markers measured across n individuals in the mapping population and aim to find ordered, connected clusters (linkage groups) $C_1, ..., C_k$. The entries of a marker feature vector x_i are individual genotypes at marker sites. In

this paper, we will explain genetic marker clustering in terms of homozygous genotypes. Hence, the n entries of a marker feature vector can take only two values A and B, or '$-$' for missing values.

Linkage groups are typically constructed by single linkage clustering based on LOD score similarities. The LOD score is the logarithm of a ratio of odds that two markers are genetically linked. A critical LOD score (*linklod* [45]) is estimated and serves as the cut-off threshold for constructing clusters from the single linkage dendrogram.

The LOD score for two markers x_i, x_j is $LOD(x_i, x_j) = \log_{10}\left(\frac{(1-\theta)^{NR}\,\theta^{R}}{0.5^{NR+R}}\right)$, where $\theta = \frac{R}{NR+R}$ is the recombination fraction, R is the number of recombinant individuals between the two markers, and NR is the number of non-recombinant individuals. Thus the LOD score is the logarithm of odds that two markers are genetically linked, under a null hypothesis of independent assortment. It is easy to show from the definition above, that the LOD score takes on a minimum value of 0 over the interval $0 \leq \theta \leq 1$ at $\theta = 1/2$ (where $R = NR$). The LOD score is symmetric about $\theta = 1/2$, taking on its maximum value of $(R + NR)\log_{10}(2)$ at $\theta = 0$ (where $R = 0$) and $\theta = 1$ (where $NR = 0$). Note that $R + NR$ is not necessarily equal to the number of individuals in the population, because the sequence data for either marker may be missing for a particular individual.

The LOD score is minimized at 0, and large positive values indicate that it is very unlikely that the markers happen to (either mismatch or) match in genotype for a large number of individuals by chance. Changes in population type (DH, RIL, F2, etc...) only affect the computation of R and NR in the LOD calculation. Thus our algorithm generalizes to more complex populations. In other words, heterozygosity will not change the fact that we depend on the LOD score to evaluate marker-marker similarity.

We point out that the fixed order of genetic markers along chromosomes is a key property of the data. Exploiting this linear, one-dimensional structure enables us to design a specialized procedure for finding linkage groups that is faster than generic clus-

tering algorithms. We use the LOD score to quickly build representative sketches of the structure of each cluster. These sketches enable us to efficiently assign each marker to its proper linkage group.

2.3 The BubbleCluster Algorithm

2.3.1 BubbleClusterAlgorithm Description

Algorithm 1 clusters in three phases: (1) perform an initial clustering $\mathcal{C}$ using high-quality markers (lines 1–17); (2) assign low-quality markers to their most likely cluster $C \in \mathcal{C}$ (lines 18–22); and finally (3) merge unrealistically small clusters with large clusters (lines 23–25). This is a coarse-to-fine approach: it relies on a good clustering of reliable high-quality data points in Phase 1 as a skeleton to assign the low-quality points in Phase 2. Such a hierarchical approach relates to theoretically well-grounded clustering techniques like core sets [46, 47] or nearest neighbor clustering [48]. In Phase 3, we identify clusters which are too small to be considered true linkage groups. We attempt to merge all such clusters with the larger clusters from Phases 1 and 2.

Our algorithm takes four parameters as input: the threshold LOD score τ, the non-missing data threshold η, a cluster size threshold σ, and an odds difference threshold c. The selection and significance of τ and η will be explained in Section 2.3.3. Briefly, τ represents the LOD score that a marker must achieve with at least one representative marker, or *sketch point*, r_j in order to join the cluster that contains r_j. Because missing data makes it impossible or very unlikely that markers with many missing entries will ever achieve a LOD score of τ, we use η to limit the number of missing entries allowed for markers included in Phase I of our algorithm. The σ input places a lower bound on the size of a cluster that the user expects could represent a true linkage group. The constant

c determines whether the odds that a marker belongs to one particular cluster is much greater than the odds that it belongs to another cluster. For ease of reference, we provide a table of these and other variables that we refer to throughout the paper in Table 2.1.

Backbone Clustering. The first, most important phase of the algorithm exploits the structure of genetic linkage groups for quickly ordering genetic markers. In Phase 1, we only process high-quality markers, that is markers with at least η non-missing entries. The algorithm establishes clusters on the fly: each incoming point is either close to, and hence assigned, to an existing cluster, or it creates a new cluster. Two clusters are merged if they are "close" in genetic distance, i.e. if points on the boundary of the clusters obtain a LOD score greater than a given threshold τ.

To avoid storing and comparing distances between a new point and all previous points, we only keep a representative sketch R_α for each cluster C_α. To create and maintain sketches, we exploit the special linear structure of the data, illustrated in Fig. 1(**b**). The resulting sketch is therefore an *ordered* list of representative points (Fig. 1(**c**)) where for every point in C_α, there is a point $r_\alpha(x)$ in R_α with $LOD(x, r_\alpha(x)) > \tau$.

For each incoming point x, we find the closest sketch point $r_{\max}$. If $LOD(x, r_{\max}) > \tau$, then x is assigned to the cluster of $r_{\max}$. Otherwise, it sprouts a new cluster (line 17). If x is added to an existing cluster, we check whether it is well represented by the current sketch, or whether we need to augment R_α. Here, we use the linearity assumption. If x is outside of the boundaries specified by R_α (the `isBdryPt()` function, line 15), we add x as a new (boundary) sketch point. If $r_{\max}$ is the only sketch point, x becomes a new sketch point automatically. If not, we compare the LOD score between x and the point $r_2 \in R_\alpha$ immediately next to r_α in the ordered list R_α, and the LOD score between r_α and r_2 as illustrated in Figures 1(**d**) and 1(**e**): if $LOD(x, r_2) < LOD(r_2, r_\alpha)$ and $LOD(x, r_2) < LOD(x, r_\alpha)$, then x extends the boundary.

When x becomes a new sketch point, it extends C_α in the linear dimension along which

we assume the sketch points to lie. It succeeds r_α and becomes a new end of R_α. Finally, we determine whether x connects two clusters (line 16) by finding the nearest sketch point $r_{\max_2}$ that is *not* in the cluster to which x was assigned. If $LOD(x, r_{\max_2}) > \tau$, then x forms a bridge $r_{\max}, x, r_{\max_2}$ between the two clusters. When merging clusters, we also merge their sketches R_α and R_β. To do so, we compare the four boundary points of R_α and R_β and append R_β to the end of R_α in the order which maintains the greatest LOD score between boundary two points, one from either cluster.

Low quality marker assignment. At the completion of Phase 1, we have an initial clustering $\mathcal{C}$ of all the high-quality data points $x \in \mathcal{H}$, along with their ordered sketches. In Phase 2 (lines 18–22), we rely on the sketches to assign the remaining low quality markers $y \in \mathcal{X} \setminus \mathcal{H}$ to one of the existing clusters. We use a simple heuristic: for each low-quality marker, we find the difference between $l_{\max_2} = LOD(y, r_{\max_2})$ and $l_{\max} = LOD(y, r_{\max})$, where $r_{\max}$ and $r_{\max_2}$ are defined as above. If this difference is greater than a threshold c, then we add y to the cluster C_α containing $r_{\max}$. Otherwise, we simply create a new, temporary *singleton* cluster containing only the point y. This choice of difference threshold means that the odds that y belongs to cluster C_α should be by a factor of 10^c greater than the odds that y belongs to any other cluster. Moreover, we only assign points to existing clusters for which we have high confidence in our assignment.

Merging small clusters with large clusters. At this stage, we rely on further assumptions about the underlying structure of our clusters, based on the following prior knowledge of true linkage groups: we know that each marker comes from exactly one linkage group, and that these groups tend to be relatively large. We attempt to merge all clusters C with $|C|$ smaller than a user-specified σ with larger clusters, by picking a random point within each small cluster and comparing its distance to all the sketch points r in large clusters. If this point is found to lie within the threshold distance of a sketch point, then we merge the small cluster with the large cluster. We can estimate σ

based on the number of markers and the number of expected linkage groups – σ is the largest cluster size that the user would consider too small to be a true linkage group.

Running time.

The BubbleCluster algorithm runs in time $O(m \log(m) + mr)$ for m markers and r sketch points. If we have chosen a threshold less than $LOD_{\max}$, which is the maximum achievable LOD score between any two markers in our dataset (and is bounded by $n \log_{10} 2$) then the number of sketch points is bounded by the number of uniquely identifiable locations in the genome, which we refer to as *bins*[1] and whose number we denote with b. For a fixed number k of chromosomes in the organism and n individuals in the mapping population, the number of bins is proportional to n and k: $b = O(nk)$ [5]. Thus $r = O(kn)$. For the organisms of interest in our work, under typical experimental conditions, the number of bins is in the thousands. In our experiments, the number of sketch points never exceeded 7% of the linkage group size, and was in fact much lower than nk.

2.3.2 Analysis

We make the following assumptions on the true underlying linkage groups $C_1^*, \ldots C_K^*$ that are roughly reasonable for real data.

A1. Separation: there exists a $\lambda_{\text{sep}} > 0$ such that for any C_α^* and any two points $x \in C_\alpha^*$, $y \notin C_\alpha^*$, it holds that $LOD(x, y) < \lambda_{\text{sep}}$.

A2. Connectedness: there exists a constant λ_{conn} with $0 < \lambda_{\text{sep}} < \lambda_{\text{conn}}$ such that for every C_α^* and each $x_1, x_2 \in C_\alpha^*$, there is a *path* of points $y_1, \ldots y_m \in C_\alpha^* \cap \mathcal{H}$

[1] Many markers may map to the same location on a chromosome.

with $LOD(x_1, y_1) > \lambda_{\text{conn}}$, $LOD(y_m, x_2) > \lambda_{\text{conn}}$ and $LOD(y_j, y_{j+1}) > \lambda_{\text{conn}}$ for all $1 \leq j \leq m$.

A3. Local linear ordering: If for three points $x_1, x_2, x_3 \in C^*_\alpha$, $LOD(x_1, x_2) > \lambda_{\text{conn}} - \delta$ and $LOD(x_2, x_3) > \lambda_{\text{conn}} - \delta$ for a $\delta > 0$, then the true order of these points is x_1, x_2, x_3 if and only if $LOD(x_1, x_3) < \min(LOD(x_1, x_2), LOD(x_2, x_3))$[2].

Lemma 2.3.1 *If $\lambda_{conn} > \tau \geq \lambda_{conn} - \delta > \lambda_{sep}$ and if A1-A3 hold, then the algorithm identifies the correct clusters for all points in $\mathcal{H}$ within one pass over the sorted data.*

Proof: First, the following invariant holds throughout and after Phase 1: any existing cluster C'_α is a subset of a true cluster, i.e., $C'_\alpha \subseteq C^*_\beta$ for some β. When a cluster is created, it consists of one point and therefore certainly is contained in a single true cluster. If a new point x gets added to C'_α, that point is within a LOD score of $\tau > \lambda_{\text{sep}}$ of $r_{\max} \in C^*_\beta$, and hence by A1, x and $r_{\max}$ must be in the same true cluster. Two clusters are merged only if there is a path $(r_{\max}, x, r_{\max_2})$ between them with a LOD score of at least τ at each hop. By A1, these clusters must therefore belong to the same true cluster.

Second, we see that if $C'_\alpha \subseteq C^*_\gamma$ and $C'_\beta \subseteq C^*_\gamma$, then $\alpha = \beta$, i.e., no true cluster is split: If C^*_γ was split, then, by A2, there would be points $y_\alpha \in C'_\alpha$ and $y_\beta \in C'_\beta$ with $LOD(y_\alpha, y_\beta) > \lambda_{\text{conn}}$. Let r_α be the point that y_α was assigned to, and r_β the point that y_β was assigned to. Then $LOD(r_\alpha, y_\alpha) > \tau$ and $LOD(r_\beta, y_\beta) > \tau$. Without loss of generality, let us assume that y_β was encountered after y_α by the algorithm, and that y_α is the closest point (highest LOD score) to y_β in C_α. Then, y_α must be a boundary point when it is added to its cluster. To see this, consider 2 cases:
(i) r_α is the only sketch point in its cluster at the time y_α is seen. In this case, y_α automatically becomes a new representative point.

[2]This assumption is supported by the fact that the recombination fraction between markers very close together on the chromosome is a reliable estimate of genetic distance [49],[50]

(ii) There are other sketch points in the cluster of r_α. By the way we merge clusters, there must be at least one sketch point r'_α with $LOD(r'_\alpha, r_\alpha) > \tau$. Since $LOD(y_\alpha, r_\alpha) > LOD(y_\alpha, r'_\alpha), LOD(y_\alpha, r_\alpha) > \tau$, and $LOD(r'_\alpha, r_\alpha) > \tau$, then by A3 y_α will be made a boundary point when it is encountered.

Since y_α is a boundary point, then when y_β is encountered, its LOD score will be highest with the sketch points r_α and y_β, with both of these LOD scores above τ. Hence, C'_α and C'_β will be merged.

■

In the presence of high missing data rates, the algorithm still provably achieves perfect precision but not perfect recall.

2.3.3 Parameters

In theory and in practice, the LOD threshold τ and non-missing data threshold η will affect both the running time and the accuracy of our algorithm. In this section, we show that τ and η are interdependent, and we explain how we choose τ and η given a population size n and a missing data rate μ. The effect of c and σ is easily explained and will be addressed at the end of the section.

Recall that a marker must achieve a LOD score above τ with a representative point in order to join that representative point's cluster, and that η limits the number of missing entries a marker vector can have in order to be included in the high-quality marker set $\mathcal{H}$. Our choices of τ and η were made to maximize the probability that each marker will be assigned to the correct cluster (set the LOD threshold τ high enough), but to also include enough points in $\mathcal{H}$ to build a reliable sketch of each cluster (set the nonmissing threshold η low enough).

Suppose that the number of observed entries n_{nm} in a marker x_i is less than the

number of observed entries in another marker x_j. As the number of nonrecombinant individuals NR in this pair of markers (x_i, x_j) approaches n_{nm}, the maximum achievable LOD score for this pair approaches $n_{\text{nm}} \log_{10} 2$ (by the definition of the LOD score in Section 2.2):

$$\lim_{NR \to n_{\text{nm}}} \log_{10} \left(\frac{\left(1 - \frac{R}{R+NR}\right)^{NR} \left(\frac{R}{R+NR}\right)^{R}}{0.5^{R+NR}} \right) = n_{\text{nm}} \log_{10} 2$$

Thus, the maximum achievable LOD score for a marker is dependent upon the number of nonmissing entries in that marker. For a LOD threshold τ, if $n_{\text{nm}} \leq \tau / \log_{10} 2$ in x_i, then line 12 will evaluate to `false` for any choice of $r_{\max}$. We want to set η high enough to prevent an overabundance of clusters from sprouting, which would necessarily raise the number of representative points and hurt efficiency – thus η should be at least $\tau / \log_{10} 2$.

We can be even more aggressive in limiting missing entries in the high-quality set, however. Given a missing rate μ, and a marker x_i with n_{nm} non-missing entries, we expect the number of non-missing entries x_i shares with any other marker will be: $E\left[\text{shared nonmissing entries}(x_i)\right] = (1-\mu) n_{\text{nm}}$. Thus if

$$(1-\mu) n_{\text{nm}} > \tau / \log_{10} 2 \Rightarrow n_{\text{nm}} > \tau / (1-\mu) \log_{10} 2$$

then we expect x_i to achieve a LOD score greater than the threshold τ with at least one other marker x_k in the high-quality set. If the structure of each cluster is indeed approximately linear, then we expect that the sketch point r nearest x_k will also achieve a high LOD with x_i, allowing x_i to be placed in the appropriate cluster. If, on the other hand, $n_{\text{nm}} \leq \tau / (1-\mu) \log_{10} 2$, we can choose to eliminate x_i from $\mathcal{H}$ because it is unlikely that x_i will score higher than τ with any sketch point.

Here the interplay between the efficiency and accuracy of our algorithm becomes

apparent. For a high missing data rate, the gap between λ_{conn} and λ_{sep} may be small, and τ must be set very high. If τ is greater than λ_{sep}, we can guarantee perfect precision, but we may eliminate so many markers from the high-quality marker set that we will not have enough markers to guarantee coverage of all clusters, resulting in low recall. Therefore, with a high missing data rate we seek to minimize the probability that we assign a marker to the wrong cluster, allowing τ to be less than λ_{sep}. Let p represent this probability:

$$p = P(LOD(x_i, x_j) > \tau | x_i \in C_i, x_j \in C_j, i \neq j)$$

By the definition of the LOD score, $p \leq 1/10^{\tau}$. Let n_{comp} be the number of LOD comparisons that we make in line 12 of our algorithm. The probability that we make no mistakes in assigning a marker to its cluster is then:

$$P(\text{no mistakes}) = (1-p)^{n_{\text{comp}}}$$

Therefore, to ensure that $P(\text{no mistakes}) > 1 - \epsilon$ for $\epsilon > 0$, we need:

$$\left(1 - \frac{1}{10^{\tau}}\right)^{n_{\text{comp}}} > 1 - \epsilon$$

$$\Rightarrow \tau > \log_{10}\left(\frac{1}{1-(1-\epsilon)^{1/n_{\text{comp}}}}\right)$$

Recall that *bins* are uniquely identifiable locations on the genetic map, and their number b is $O(kn)$ for k linkage groups and n individuals in the mapping population. An upper bound on n_{comp} is thus $\binom{b}{2}$, corresponding to the grossly pessimistic situation where every bin is represented by a representative point, and the LOD score must therefore be evaluated once for every bin pair. Although this is a worst-case scenario, we can use this upper bound to estimate τ given an $\epsilon > 0$.

Our selection of τ and η is thus a balancing act, where we want to ensure that τ is high enough to guarantee a high $P(\text{no mistakes})$, but at the same time is not so high that a large fraction of our data would be excluded from Phase 1, which we rely on to build a sketch of each cluster. For example, if we are given a population size of $n = 300$, we expect $k = 10$ linkage groups, and we want to achieve $P(\text{no mistakes}) > 0.99$, then $\tau > \log_{10}\left(1/1 - 0.99^{1/\binom{3000}{2}}\right) = 8.6509$ would achieve perfect precision with 99% confidence. Given a missing data rate $\mu = 35\%$, we require $n_{\text{nm}} > 8.6509/(0.65)\log_{10} 2 = 44.2117$, so we would set η to 44. If $\mu = 65\%$, then by the same calculation we would require $\eta \geq 82.1076$. If this choice of η excludes too many markers from $\mathcal{H}$, we might choose to either raise ϵ or allow less-than-perfect precision (achieve a low $P(\text{number of mistakes} > \text{constant})$) in exchange for greater coverage of the linkage groups by the markers in $\mathcal{H}$. In practice, we achieve extremely high precision as well as recall using these crude estimates.

Although c and σ also influence the resulting cluster quality, their effect is quite obvious. Higher c values prevent markers with many missing entries to be assigned to any cluster, resulting in more small clusters in the output. These small clusters can be left out of the final genetic map or assigned to larger clusters by more careful analysis by the user. Similarly, high values of σ can cause unnecessary comparisons to be made between large clusters that already represent linkage groups, while extremely small values may miss the opportunity to merge small clusters with larger clusters.

2.3.4 Related Work

Several computational tools exist for the construction of genetic linkage maps, as explained in the survey by Cheema and Dicks [5]. Since then, OneMap [51] and LepMAP [52] have also been proposed. All these tools, without exception, perform all-pairs comparisons among markers, making them unsuitable for datasets with millions

of markers. Structural clustering methods that have been applied to genetic mapping include connected components [53, 54, 39] and single linkage clustering [55]. Differently from single linkage, we construct and merge clusters on the fly, requiring only one pass through the data after sorting.

General compressed representations for clustering problems have been addressed by core sets [46, 47], and by hierarchical re-clustering ideas for streaming and distributed clustering [56, 57]. As opposed to general sampling techniques, we extract a problem-specific representative core set deterministically within one pass, and exploit the specific structure of the marker data.

Our algorithm maintains an ordering of the dataset that is similar in spirit to the OPTICS [58], DBSCAN [59], or BIRCH [60] algorithms. However, our algorithm is not density based. Applying density-based approaches to genetic marker data would be difficult if not impossible, due to the lack of a distance metric with which to compute inter-marker distances. The challenge is converting the LOD similarity score into a valid distance metric which respects the triangle inequality. We cannot use density-based approaches which rely on the notion of an "$\epsilon-$neighborhood" around data points in order to find a dense region of the space in which the data lie.

Our algorithm uses several representative points to provide an accurate coverage of the underlying cluster. In that sense, our approach is closest to the CURE algorithm [61], which also maintains representative points. The specific insight we draw from the genetic mapping problem enables our algorithm to maintain a better performance bound than CURE's $O(m^2 \log m)$ bound, and allows us to prove correctness with mild assumptions.

2.4 Experimental Evaluation

We compare our algorithm to two popular genetic mapping tools: JoinMap and MSTMap. We also provide a comparison with PIC, a spectral clustering approach [21]. Most of the experiments were run on Neumann, a quad-core server with AMD Opteron 8378 Processors running at 2.4GHz. Because JoinMap requires the Windows OS, experiments with JoinMap were performed on a Windows desktop with Intel 2.93GHz Core 2 Duo processors. Our code was written in C++ and compiled with gcc 4.4.7. All experiments are single threaded and use a single core.

2.4.1 Data

We evaluate BubbleCluster on both real and synthetic datasets. The first dataset, *barley*, consists of 65,357 genetic markers from a population of 90 individuals with 20% missing data. This species of barley has 7 true linkage groups. The second, larger *switchgrass* dataset of 548,281 genetic markers comes from a population of 500 individuals (with some replicated individuals for better coverage), with 65% missing data and 18 true linkage groups. Due to its size, previous clustering efforts on this data focused only on the 113,326 highest-quality markers. We cluster both the 113K subset of markers and the complete 548K dataset in our experiments to demonstrate the scalability of our algorithm. Our third timing result on real data is for the grand-challenge *wheat* genome, containing 1,582,606 markers from a population of 88 individuals and 21 true linkage groups with 39% missing data entries. We do not report accuracy results on wheat because single-linkage clustering failed to provide a golden standard result to compare to after a week of computation given all our resources.

For scaling studies, we rely primarily on synthetic data generated by the Spaghetti software, which simulates genetic marker data with real-life complications [62]. In par-

ticular, we created datasets for a range of missing data rates, from 0 to 65%. We varied the number of markers from 12.5K to 400K, doubling the size at each increment. The population size was fixed at 300, the sequencing error rate at 0.1%, and the number of linkage groups at 10 in all experiments.

2.4.2 Evaluation Metric

We use the overall F-score to evaluate the quality of each clustering. The F-score ranges from 0 (no correspondence) to 1 (perfect match), and evaluates a test cluster C_i with respect to a "golden standard" cluster G_j in terms of precision and recall [63]. Formally, if $G_j \in \mathcal{G}$ is a golden standard cluster, then the F-score for a test cluster C_i with respect to G_j is defined as: $F(G_j, C_i) = \frac{2p_{ij}r_{ij}}{r_{ij}+p_{ij}}$, for recall $r_{ij} = \frac{|C_i \cap G_j|}{|G_j|}$ and precision $p_{ij} = \frac{|C_i \cap G_j|}{|C_i|}$. The *overall* F-score is a normalized, weighted sum of the F-scores for each golden standard cluster $G_j \in \mathcal{G}$: $F(\mathcal{G}, \mathcal{C}) = \frac{1}{m} \sum_{j=1}^{k} |G_j| \max_{i=1\ldots l} F(G_j, C_i)$, where k is the number of true clusters, l is the number of test clusters, and m is the total number of datapoints. "True" clusters are generated directly in simulated data experiments; for real data, if assumption A1 holds, then single linkage clustering will provably find the correct clusters given a threshold $\tau > \lambda_{\text{sep}}$. We thus rely on the outcome of single linkage clustering to measure accuracy on real data.

2.4.3 Results

Table 2.2 summarizes the running time and F-score on the real data sets. A LOD threshold of $\tau = 8$ with nonmissing data threshold $\eta = 66$ was used to cluster the 65K barley dataset; for switchgrass, we used thresholds $\tau = 20$ and $\eta = 132$; for wheat, we fixed $\tau = 9$ and $\eta = 44$. The c parameter was set to 5, and σ to 100, in all experiments. Our selection of τ and η was based on two objectives, as explained in Section 2.3.3: (1)

maximize the probability that each marker is assigned to the correct cluster, but at the same time (2) ensure that Phase 1 contains sufficiently many markers to build a reliable sketch of each cluster. No mapping tool we know of, including the popular MSTMap or JoinMap, has been successful in clustering genetic marker datasets at this scale.

Table 2.2 demonstrates that we achieve very accurate clusters in $O(m \log m)$ time for m markers, a significant improvement over the $O(m^2)$ algorithms used by other genetic marker clustering tools. We emphasize that these datasets come from real-world sequence data, where missing data entries do not have a simple known distribution. Nonetheless, our algorithm recovers the linkage groups with both precision and recall above 97%. We omit comparisons with MSTMap on all but the smallest dataset of 64K markers, where it took MSTMap almost a *week* to ultimately place all the markers in the same cluster.

The clustering accuracy of the recently sequenced wheat genome, with 1.582 million markers, has been independently validated in a recent study [64]. In summary, our algorithm achieves very high accuracy at fast running times.

Table 2.3 underscores our ability to outperform existing genetic clustering methods as well as more general clustering methods applied to synthetic genetic marker data. Even on relatively small datasets, where JoinMap and MSTMap complete the clustering stage in a reasonable amount of time, we achieve identical F-scores as those methods, but within a fraction of the time. In fact, the results are slightly biased in favor of JoinMap. Although this tool will automatically construct the single-linkage dendrogram from an input data matrix, the user must select which dendrogram edges to cut in order to produce the final clusters. We selected the edges that resulted in the best clustering based on our prior knowledge of the simulated linkage groups. The time reported is only the time for generating the JoinMap dendrogram.

The best run out of several re-runs with different counts of pseudo-eigenvectors for the PIC algorithm is reported in Table 2.3. The clustering was performed using k-

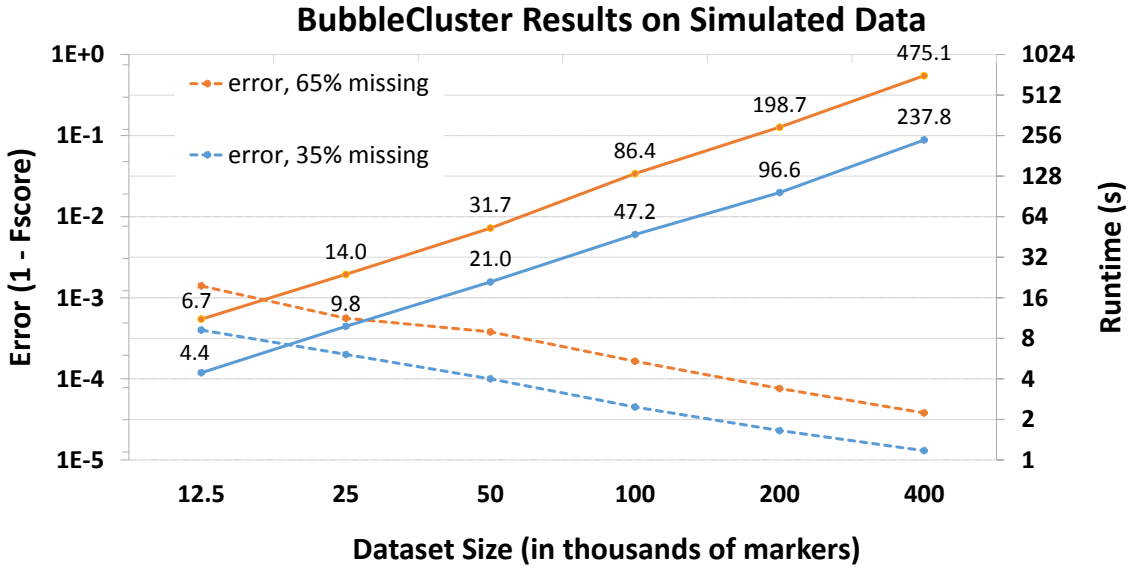

Figure 2.2: Average errors ($1 - F$-score, left axis) and runtimes (right axis) for increasing dataset sizes.

means++ [26] on the data projected onto the two-dimensional space spanned by two pseudo-eigenvectors. This procedure performed empirically the best. We include the running time of PIC given the similarity matrix as input, and indicate the $O(m^2)$ time required to construct this matrix in parentheses. The inability of PIC to produce competitive results in terms of clustering quality motivates the need for a more application-specific approach in this domain. We point out the drawbacks in applying general clustering methods to a problem with missing data, a similarity function that cannot be expressed as an inner product, and an underlying structure that can only be exploited if the application-specific problem is well understood.

Our ability to scale, while simultaneously making use of more data availability, is demonstrated in Fig. 2.2. Here, we increase the size of our synthetic dataset from 12.5K to 400K genetic markers. We report the clustering quality for both 35% and 65% missing data in terms of the errors we make; the error is reported in terms of ($1 - F$-score) for each (dataset size, missing data rate) pair (left-hand axis of Fig. 2.2). The running times

for the same data are plotted with respect to the right-hand axis of Fig. 2.2. Both the running time and errors are averages of five trials on each dataset size. We make two points about these empirical results: (1) the error we make in clustering decreases linearly, in almost exact correlation with the size of the data, and (2) the running time increases with $O(m \log m)$, promising reliable performance up to almost half a million markers, even with an enormous amount of missing data. Comparing Table 2.2 with these results, we believe the behavior of our algorithm in these experiments is predictive of its performance in the real world.

Lastly, we make a note on the impact of our choices of thresholds on the cluster quality and the running time. Tables 2.4 and 2.5 capture the behavior of BubbleCluster on a 200K simulated dataset with 300 individuals with fixed missing data rates of 65% and 35%, respectively. In the top half of Table 2.4, we fix η at 66 and vary τ. With this choice of η, any marker that can achieve a maximum LOD score of at least 20 will be included in the high-quality set $\mathcal{H}$. The running time increases with increasing τ, as expected; a higher τ will cause more clusters and sketch points to be created, increasing the number of LOD comparisons the algorithm needs to make. The F-score also increases up to $\tau = 8$, then drops off slightly for increasing values of τ. This is due to Phase III of our algorithm – at $\tau > 8$, small clusters are created that cannot be merged with any large existing cluster because no marker within the small clusters achieves a high enough LOD score with any of the large clusters' sketch points.

In the bottom half of Table 2.4, we reverse the roles of η and τ. Here, τ remains fixed at 8 and we increase η from 66 (22% observed entries) to 149 (49.67% observed). In parentheses we show what we call the *self-lod* of a marker for the given value of η, which is the LOD score a marker would achieve with itself if it had $n - \eta$ missing entries, i.e. the maximum achievable LOD score for a marker with η observed entries. We see that at $\eta = 66$, every marker is included in the high-quality set $\mathcal{H}$, and the F-score is

very high. As we increase η, more markers are excluded from $\mathcal{H}$ and the F-score suffers. Note that at $\eta = 99$, more than 50K markers are excluded from $\mathcal{H}$, resulting in poorer coverage of the clusters with sketch points and a greater chance that low-quality markers will not achieve a high LOD score with any sketch point. These low-quality markers are left out of large clusters, decreasing the recall in the $F-$score. As we increase η to exclude a majority of the markers, the F-score drops off dramatically. These results show that while we attempt to eliminate very "low-quality" markers from our dataset, a high enough LOD threshold allows us to include many markers with high amounts of missing data in our "high-quality" set $\mathcal{H}$, producing very accurate clusters.

Table 2.5, with analogous results to Table 2.4 for 35% missing data, tells a similar story. The running time increases with increasing τ and fixed η. However, here we see a much wider range of τ values will give accurate results very quickly. We can afford to set τ to a much higher value than in the case of 65% missing data, allowing η to be low enough to include all the markers in our dataset in $\mathcal{H}$ for higher F-scores.

2.5 Discussion

Current approaches to genetic mapping were designed for a small data setting, and use algorithms that scale quadratically in the number of markers. We propose an approach that exploits the underlying linear structure of chromosomes to avoid expensive comparisons between (quadratically) many pairs of markers. The resulting linkage groups (i.e., marker clusters) are highly concordant with computationally expensive quadratic calculations, but our improved scaling allows far denser maps to be constructed with minimal computation.

After the formation of linkage groups, the next step in constructing a high quality genetic map is inferring the detailed ordering of markers along chromosomes. Since our

method takes into account the linear structure of chromosomes from the start, the result is an approximate marker ordering that is an excellent starting point for detailed marker order by simulated annealing or other methods that explore short-range perturbations of our approximate ordering. In fact, the sketch point order found by our algorithm for the barley dataset (Sec. 2.4.1), was highly correlated with the true marker order in barley linkage groups: for 6 out of 7 groups, the Spearman Rank Correlation Coefficient ρ was above 0.9. We are currently working on an efficient ordering algorithm that can use the results of BubbleCluster to infer missing data and to quickly order markers.

Our algorithm can significantly speed up current genetic mapping efforts on large datasets. Though the ordering phase of genetic mapping has been shown to be NP-hard, efficient heuristic algorithms have been proposed to quickly order markers within a linkage group [39]. BubbleCluster eliminates the bottleneck in current genetic mapping tools in the big data setting. An important application of our method is in the efficient construction of ultra-dense genetic maps for large and complex genomes that are filled with repetitive sequences that frustrate genome assembly but do not limit the number of genetic markers. The most economically important of these genomes are various grasses, including crops grown for food (e.g., barley and wheat, whose genome sizes are two- to seven-fold larger than the human genome) or as biofuel feedstocks (e.g, switchgrass and miscanthus, polyploids that contain multiple, subtly different copies of a basic genome).

Algorithm 1: BubbleCluster Algorithm

Inputs: $\mathcal{X} = \{x_1 \dots x_M\}$, τ, η, c, σ

$\mathcal{C} \leftarrow \emptyset$; $\mathcal{R} \leftarrow \emptyset$; `// Lists of cluster and representative sets`

sort $\mathcal{X}$ by increasing missing data;

$\mathcal{H} = \{x_i \in \mathcal{X} |$ nonmissing$(x_i) > \eta\}$;

if $|\mathcal{H}| == 0$ **then** return $\mathcal{C}, \mathcal{R}$;

for *point* $x \in \mathcal{H}$ *in sequence* **do**

  if $\mathcal{R} = \emptyset$ **then**

    define new cluster C_α: $C_\alpha \leftarrow \{x\}$;

    define new rep. set R_α: $R_\alpha \leftarrow \{x\}$;

    $\mathcal{C} \leftarrow \mathcal{C} \cup C_\alpha, \mathcal{R} \leftarrow \mathcal{R} \cup R_\alpha$;

  else

    $r_{\max} = \underset{r}{\text{argmax}}\ LOD(x, r)$ s.t. $r \in R_\alpha \in \mathcal{R}$;

    if $LOD(x, r_{\max}) > \tau$ **then**

      $r_{\max_2} = \underset{r \notin R_\alpha}{\text{argmax}}\ LOD(x, r)$;

      assign x to cluster C_α: $C_\alpha \leftarrow C_\alpha \cup \{x\}$;

      if `isBdryPt` (x, R_α) **then** add x to the correct end of R_α ;

      if $LOD(x, r_{\max_2}) > \tau$ **then** `MergeRs` (R_α, R_β); `MergeCs` (C_α, C_β) ;

    else set up new cluster: $\mathcal{C} \leftarrow \mathcal{C} \cup \{x\}, \mathcal{R} \leftarrow \mathcal{R} \cup \{x\}$;

for $y \in \mathcal{X} \setminus \mathcal{H}$ **do**

  $r_{\max} = \underset{r}{\text{argmax}}\ LOD(y, r)$ s.t. $r \in R_\alpha \in \mathcal{R}$; $r_{\max_2} = \underset{r \notin R_\alpha}{\text{argmax}}\ LOD(y, r)$;

  $l_{\max_2} = LOD(y, r_{\max_2})$; $l_{\max} = LOD(y, r_{\max})$;

  if $(l_{\max} - l_{\max_2}) > c$ **then** $C_\alpha \leftarrow C_\alpha \cup \{y\}$;

  else set up new cluster: $\mathcal{C} \leftarrow \mathcal{C} \cup \{y\}, \mathcal{R} \leftarrow \mathcal{R} \cup \{y\}$;

for *all* C *in* $\mathcal{C}$ *s.t.* $|C| < \sigma$ **do**

  pick a $c_j \in C$;

  if $LOD(c_j, r_k) > \tau$ *for any* r_k *in any* $R_\alpha \in \mathcal{R}$ **then** `MergeCs` (C, C_α) ;

τ	LOD threshold
η	non-missing data threshold (applied to marker vector entries)
σ	cluster size threshold
c	odds difference threshold
μ	missing data rate in the dataset
n	population size (number of individuals in the mapping population)
k	number of clusters
ϵ	error tolerance for misassignment of markers to clusters
n_{nm}	number of non-missing entries in a particular marker vector
r	number of representative points aka sketch points
b	number of bins, i.e. unique locations on the genetic map, always $O(nk)$
$\mathcal{H}$	high-quality set, defined as the set of markers with $n_{\text{nm}} > \eta$

Table 2.1: List of parameters/variables used

Dataset	Markers	BubbleCluster	
		F-score	Time
Barley	64K	0.9993	15 sec
S-grass	113K	0.9745	8.9 min
S-grass	548K	0.9894	1.9 hrs
Wheat	1.582M	N/A	1.22 hrs

Table 2.2: Clustering performance on Barley, Switchgrass, and Wheat from the Joint Genome Institute using BubbleCluster. MSTmap and JoinMAP are unable to cluster data sets at this scale.

Clustering	12.5K Markers		25K Markers	
	F-Score	Time	F-Score	Time
JoinMAP	0.9996	14 min	0.9998	46 min
MSTMap	0.9996	4.5 min	0.9998	20 min
PIC	0.4702	11 sec +(4 min)	0.6078	44 sec +(16.5 min)
Bubble	**0.9996**	**6 sec**	**0.9998**	**15 sec**

Table 2.3: Performance comparison of clustering algorithms using synthetic Spaghetti with 35% missing data and 0.1% error rate. All experiments ran on Neumann (Linux/AMD), except JoinMap ran on the Windows machine. The parenthesized PIC algorithm number is preprocessing time for pairwise calculations.

200K Markers, 300 individuals, 65% missing, $\eta = 66$						
τ	5	6	7	8	9	10
F-Score	0.1894	0.5240	0.9261	0.9999	0.9998	0.9988
Time (s)	82.5	124	155	183	242	307
200K Markers, 300 individuals, 65% missing, $\tau = 8$						
η (self-lod)	66 (20)	83 (25)	99 (30)	116 (35)	132 (40)	149 (45)
F-Score	0.9999	0.9982	0.9004	0.6169	0.4986	N/A
Time (s)	183	190	180	101	42.3	N/A
# markers with $n_{\text{nm}} > \eta$	200,000	199,205	148,914	16,551	99	0

Table 2.4: F-scores & running times for varying choices of η & τ on a 200K dataset with 65% missing data.

200K Markers, 300 individuals, 35% missing, $\eta = 132$						
τ	5	10	15	20	25	30
F-Score	0.6225	0.9999	0.9999	0.9999	0.9999	0.9999
Time (s)	48.6	67.0	70.9	82.0	106	170
200K Markers, 300 individuals, 35% missing, $\tau = 20$						
η (self-lod)	132 (40)	166 (50)	172 (52)	179 (54)	186 (56)	192 (58)
F-Score	0.9999	0.9999	0.9992	0.9930	0.9610	0.8948
Time (s)	82.0	84.6	82.7	83.0	81.7	82.0
# markers with $n_{\text{nm}} > \eta$	200,000	199,920	199,296	193,701	169,648	124,263

Table 2.5: F-scores & running times for varying choices of η & τ on a 200K dataset with 35% missing data.

Chapter 3

Data reduction for genetic mapping

We present a fast and accurate algorithm for reducing large-scale genetic marker data to a smaller, less noisy, and more complete set of *bins*, representing uniquely identifiable locations on a chromosome. Our experimental results on real and synthetic data show that our algorithm runs in near-linear time, allowing for the analysis of *millions* of markers. Our algorithm reduces the problem scale while preserving accuracy, making it feasible to use existing genetic mapping tools without resorting to complex, time-intensive preprocessing methods to filter or sample the original data set. Additionally, our approach also decreases the uncertainty in genotype calls, improving the quality of the data. Preliminary results demonstrate that existing methods for marker ordering designed for the small scale settings perform with equivalent accuracy when given our reduced bin set as input.

3.1 Introduction

We make the following three important contributions to genetic mapping: (1) We efficiently process genetic marker data sets orders of magnitude greater than any known genetic mapping tool in nearly-linear time, reducing the data to a size that can be easily managed by traditional mapping tools. (2) We show that our method discovers the

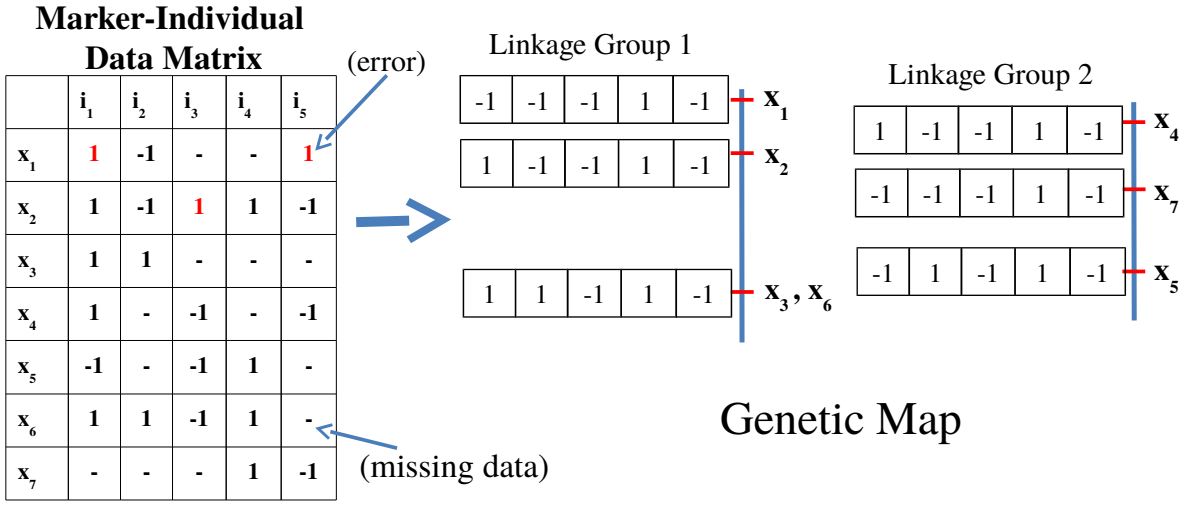

Figure 3.1: Producing a genetic map from a data matrix of homozygous genetic markers.

fundamental resolution of genetic marker data nearly perfectly, even for very noisy data sets with many missing values. (3) By reducing genetic marker data to its fundamental resolution, we preserve information relevant to genetic map construction while eliminating errors and uncertainty.

Recall that genetic mapping is generally described as a three-step process[5]: (1) Cluster genetic markers into sets that represent *linkage groups*, ideally in one to one correspondence with chromosomes. (2) Order the markers within each linkage group. (3) Determine the genetic spacing between ordered markers within each linkage group. Figure 3.1 illustrates the outcome of genetic mapping for a data matrix of $m = 7$ *homozygous* genetic markers from a *mapping population* of $n = 5$ individuals. Homozygosity guarantees that each marker can only take on one of two values for a given individual. Each marker x_j is given as a row in the data matrix, with the number of columns equal to mapping population size. Homozygous markers are abundant for many population types used in genetic mapping (e.g. doubled-haploid (DH) and advanced recombinant inbred line (RIL) populations) [5, 39, 65]. Each column represents the homozygous state of a genetic marker, +1 or -1, or 0 for missing data, for each individual. The end-product of

the genetic mapping pipeline is a blueprint of the genome, showing which markers belong to which linkage group, and within each linkage group, genetic map locations correspond to n-dimensional vectors that indicate genotypes at these locations. Note that, as is the case for x_3 and x_6 in the figure, more than one marker can map to the same location.

Figure 3.1 also alludes to the computational difficulties associated with generating a high-quality, large-scale genetic map. First, large amounts of missing data make it difficult to cluster markers into linkage groups using simple similarity metrics. Second, even small genotyping error rates cause some genotypes to appear as *opposite* of their true value (i.e. +1 flips to -1, or visa-versa, in the data matrix), adding a layer of difficulty for analyzing marker linkage patterns. Third, as the amount of data (rows in the data matrix) grows into the millions, traditional polynomial-time algorithms that were designed for small-scale genetic mapping are not sufficiently fast to process the data. Errors and missing data values in large data sets have been shown to dramatically affect the quality of the genetic map produced by existing tools ([39]). Therefore, our algorithm not only improves the efficiency of the mapping process by reducing the data set size, but by reducing errors and missing values, has the potential to significantly improve the quality of the final map.

In this work, we aim to reduce genetic mapping data to its fundamental resolution, which is independent of the data set size. Differently from a map's *density*, which is limited by the number of genetic markers, the *resolution* of a map depends on the number of individuals in the mapping population. We propose an algorithm that reduces the scale of genetic marker data to its fundamental resolution, thereby reducing noise and enabling classic bioinformatic algorithms to be efficiently applied to new large-scale data sets with increased robustness. By running experiments on synthetic data, as well as real data consisting of barley and the highly complex wheat genome, we demonstrate that our approach can perform data reduction in near-linear time. Additionally, we show

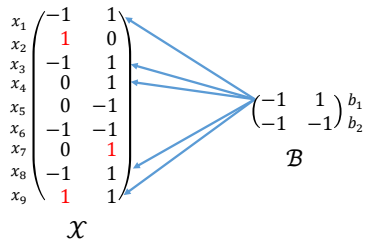

Figure 3.2: Input data illustration. Here, $\mathcal{B}$ is a set of 2 distinct 2-vectors which generated the 9 markers $x_1, \ldots, x_9$. The arrows indicate the markers that belong to bin_1, corresponding to bin vector b_1. The rest of the x_j's belong to bin_2. Red entries represent genotyping errors and 0's indicate missing data.

that our novel algorithm preserves high accuracy compared with the gold standard, and produces a reduced bin set whose ordering is almost identical to the solution obtained by MSTMap [39] given the exact bins as input.

3.2 Problem Definition

We assume that M homozygous markers are given to us as input. Thus each marker x_j is a vector with each entry $x_j(i)$ represented by values +1 or -1, or 0 for missing data. Genetic maps are built by computing probabilities of linkage between genetic markers, based on similarities between genotypes of individuals in the mapping population. In the absence of errors, differences in genotypes for a single individual at two different markers are based solely on the concept of *recombination*. Typically, one to two recombination events occur per chromosome per individual [49, 50]. For two markers x_j, x_l on the same chromosome, if no recombination event occurs between them, then the vectors will be *indistinguishable* (i.e. $x_j = x_l$) in the data matrix.

This insight is the key to our argument that a fundamental resolution exists for any fixed population size, independent of the data set size, and leads to the effectiveness of our algorithm. The number of uniquely identifiable locations on a genetic map, referred to *bin vectors*, is limited by the number of recombination events that occur in the population,

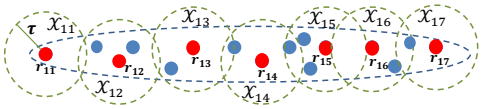

Figure 3.3: Visualization of our BubbleCluster algorithm. Each linkage group is represented as a cluster spanned by a set of sketch (red) points $r_{\kappa q}$, such that each marker x_j (blue) is within a threshold LOD score τ of at least one sketch point. $\mathcal{X}_{\kappa q}$ represents the sets of markers closest to sketch point $r_{\kappa q}$.

which in turn is limited by the number of chromosomes (k) in the genome and the number of individuals (n) in the mapping population. The number of recombination events is $O(kn)$, with the constant in the big-O notation typically close to 1 [66, 5].

The idea that many markers may map to the same location in a genetic map is well known in the bioinformatics community [39, 5, 45, 55, 67]. The novelty in our approach is to frame genetic mapping as a data reduction problem, and to leverage large amounts of data to discover the bin vectors that establish the fundamental resolution size.

The data reduction problem is stated as follows. The goal is to find the set $\mathcal{B}$ of distinct *bin vectors* $b_1, \ldots, b_q \in \{-1, 0, 1\}^{1\times n}$ that generated the data $\mathcal{X} \in \{-1, 0, 1\}^{M\times n}$. We assume a fixed error rate ϵ in the data. Thus, if b_k generated x_j, then $x_j(i) = b_k(i)$ with probability $1 - \epsilon$ and $x_j(i) \neq b_k(i)$ with probability ϵ. We further assume a fixed missing rate μ, giving an independent probability to the event that $x_j(i) = 0$. Finally, we assume that $M \gg n$. In addition to the bin vectors b_q, we seek to identify the set bin_q of markers generated by each b_q. Figure 3.2 gives a tiny example of 9×2 input data $\mathcal{X}$, generated from the 2×1 bin vectors $b_1, b_2 \in \mathcal{B}$. Here the markers x_1, x_3, x_4, x_8 and x_9 comprise the set bin_1, and the remainder of the markers belong to bin_2.

Note that the number of genetic markers can be orders of magnitude larger than the number of bins. In other words, for large data sets, multiple markers $x_j \in \mathcal{X}$ will necessarily map to the same location $b_k \in \mathcal{B}$ in the genetic map by the Pigeonhole Principle. Thus, even with high missing data rates and realistic error rates, large amounts of data allow us to eliminate much of the uncertainty in bin vector values, as most markers

x_j in the same bin will agree in genotype values for all non-missing entries $x_j(i) \neq 0$. The challenge is to reduce the M input markers to $O(kn)$ bins.

Our previous work presented the BubbleCluster algorithm [68] (described in Chapter 2), which demonstrated that large sets of markers can be efficiently partitioned into clusters by forming a linked chain of *sketch points* that span each cluster. Markers that achieve a LOD score of τ or greater with a particular sketch point are genetically linked with high probability. A LOD score between two homozygous markers x_j and x_k is given by the formula: $LOD(x_j, x_k) = \log_{10}\left(\frac{\left(\frac{o}{\eta_{jk}}\right)^{o}\left(\frac{\eta_{jk}-o}{\eta_{jk}}\right)^{\eta_{jk}-o}}{2^{\eta_{jk}}}\right)$ where η_{jk} is the number of entries i for which both $x_j(i) \neq 0$ and $x_k(i) \neq 0$, and o is the number of entries i for which $x_j(i) \neq 0$, $x_k(i) \neq 0$, and $x_j(i) \neq x_k(i)$. Thus η is the number of non-missing entries shared by both x_j and x_k, and o is the number of those non-missing entries for which x_j and x_k disagree in genotype. BubbleCluster is similar in spirit to density-based clustering, but with a threshold similarity score τ used to determine when to link points to the same cluster, instead of applying a threshold on the density surrounding a point before linking it with others in the same cluster. BubbleCluster is visualized in Figure 3.2, where markers are assigned to clusters, but also partitioned within clusters into groups $\mathcal{X}_{\kappa q}$. Each $\mathcal{X}_{\kappa q}$ represents the set of markers with high probability of genetic linkage with a particular sketch point $r_{\kappa q}$, corresponding to the qth sketch point in cluster κ. We use the sets $\mathcal{X}_{\kappa q}$ as a starting point, or coarse clustering, to obtain a more refined clustering that will enable us to discover the bin vectors in our data set.

3.3 Data Reduction Algorithm Description

We now present our algorithm that follows a course-to-fine clustering approach. The algorithm first partitions markers into linkage groups, then obtains a coarse clustering

within each linkage group into sets $\mathcal{X}_{\kappa q}$, and uses these sets to estimate the error rate ϵ in our data set. Then, the estimated ϵ is used to recursively bisect each set $\mathcal{X}_{\kappa q}$ until the fraction of errors within a set is close to our error estimate for each individual i. Each set of markers returned by our algorithm represents a bin bin_k, and the bin vector values $b_k(i)$ are assigned intuitively, becoming +1 if for a majority of the markers $x_j \in \text{bin}_k$, $x_j(i) = +1$, and -1 otherwise. We expand on the details of our algorithm below.

3.3.1 Estimating the Error Rate

Algorithm 2 describes the process used to derive bins from a set of markers $\mathcal{X}$. First, we rely on Phase I of our previously published BubbleCluster algorithm to partition the marker set $\mathcal{X}$ into clusters $\mathcal{C}$ along with respective sketch point sets $\mathcal{R}$ (Line 1). The sketch point sets obey the property that for every marker x_j in a linkage group cluster C_κ, x_j is within a threshold LOD score τ of at least one sketch point r_p in the sketch point set R_κ corresponding to C_κ. Furthermore, every sketch point r_p is within τ of at least one other sketch point in R_κ (see Figure 3.2).

The output sketches $R_1, \ldots, R_\kappa$ are used to further partition $\mathcal{X}$ into sets $\mathcal{X}_{\kappa q}$ (Line 5). In this step, we re-assign each marker x to the sketch point r that achieves the highest LOD score with x, breaking ties randomly. Thus, every set $X_{\kappa q}$ represents a set of markers that are all highly likely to be close together on the chromosome. In the next step of our algorithm, we use this assumption to estimate the genotyping error rate in our data and to find the bins.

To estimate the error rate ϵ, we assume that ϵ is a fixed constant, and that for every marker x_j, the probability that the ith individual entry $x_j(i)$ is assigned to the incorrect genotype is independently ϵ. Thus, the distribution of errors in the data set follows a Bernoulli distribution. Therefore, assuming one recombination per individual

Algorithm 2: Algorithm FindAllBins

Inputs : $\mathcal{X} = \{x_1 \dots x_M\}$=set of markers
Output: bins=a set of sets of markers b_k, where each b_k represents a bin,
$\mathcal{F}$=set of bin vectors corresponding to each bin set

1 $(\mathcal{R}, \mathcal{C})$ = BubbleCluster$(\mathcal{X}, \tau, \eta)$;
2 mles $= \emptyset$; $\mathcal{F} = \emptyset$;
3 **for** $\kappa = 1 \dots |\mathcal{R}|$ **do**
4 **for** $q = 1 \dots |R_\kappa|$ **do**
5 $\mathcal{X}_{\kappa q} = \{x \in \mathcal{X} | q = \max_l LOD(x, r_l \in R_\kappa \in \mathcal{R})\}$;
6 $\text{mles}_{\kappa q} = \text{median}(\text{getMLEs}(\mathcal{X}_{\kappa q}))$;
7 $\hat{\epsilon} = \left(\sum_{\kappa=1}^{|\mathcal{R}|} \sum_{q=1}^{|R_\kappa|} |\mathcal{X}_{\kappa q}| \text{mles}_{\kappa q}\right) / \sum_{\kappa=1}^{|\mathcal{R}|} \sum_{q=1}^{|R_\kappa|} |\mathcal{X}_{\kappa q}|$;
8 **if** $\hat{\epsilon} == 0$ **then**
9 $\hat{\epsilon} = 0.00001$;
10 $\alpha = \hat{\epsilon}(1 - \mu)M$;
11 bins $= \emptyset$;
12 **for** $\kappa = 1 \dots |\mathcal{R}|$ **do**
13 **for** $q = 1 \dots |R_\kappa|$ **do**
14 bins = bins $\cup$ RecursiveBisectionBins$(\mathcal{X}_{\kappa q}, \hat{\epsilon}, \alpha)$;
15 **for** $j = 1 \dots |\text{bins}|$ **do**
16 $\mathcal{F} = \mathcal{F} \cup$getBinGenotypes(bin$_j$);
17 return ($\mathcal{F}$,bins);

per chromosome, the maximum likelihood estimate (MLE) of the error rate for markers in the same bin, at one individual i, is $\frac{o}{\eta}$, where η is the number of markers for which the genotype is defined (not missing, $x_j(i) \neq 0$), and o represents the number of recombinant genotypes (number of markers for which the genotype is opposite that of the bin vector, $x_j(i) \neq b(i), x_j(i) \neq 0$). Having no prior information, the `getMLEs` function on Line 6 returns a vector of MLE error estimates for each individual, where each MLE is $\frac{o}{\eta}$ for that individual, simply assuming that o is number of minimally-occurring genotypes for i.

We can view the coarse-grained clustering of markers into sketch point sets $\mathcal{X}_{\kappa q}$ as a grouping containing significantly fewer bins than the original data. However, it is a challenging problem to partition each $\mathcal{X}_{\kappa q}$ into bins. We can nonetheless assure that no $\mathcal{X}_{\kappa q}$ contains markers more than $n/2$ bins by setting the LOD threshold τ in the

BubbleCluster algorithm high enough and thus the *median* MLE error rate estimate will approach the true error rate for every $X_{\kappa q}$ as the amount of non-missing data in each $\mathcal{X}_{\kappa q}$ increases. After collecting the MLE's for each individual i in each set $\mathcal{X}_{\kappa q}$ (Line 6), the error estimate $\hat{\epsilon}$ is set to a weighted average of the median MLE's from each sketch point bubble $\mathcal{X}_{\kappa q}$, with weights equal to the normalized sizes of $\mathcal{X}_{\kappa q}$ (Line 7). Weights thus give $\mathcal{X}_{\kappa q}$'s with more data increased influence over the selection of $\hat{\epsilon}$.

Because the errors follow a Bernoulli distribution, we use the Beta distribution, its conjugate prior, to quantify our level of confidence in $\hat{\epsilon}$, our estimate of ϵ. Recall that a Beta distribution takes two parameters, α and β, and has mean $\frac{\alpha}{\alpha+\beta}$ and variance $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$. Thus, as α and β increase, the variance decreases, effectively placing more confidence in the mean. In Line 10, we consider α to be the prior belief in the number of errors that occur in the data. We set the α parameter to be the number of markers in the data set multiplied by the estimated error rate $\hat{\epsilon}$ and by one minus the missing rate μ. Multiplying the number of markers by one minus the missing rate gives the total non-missing entries entries $x_j(i) \neq 0$ that are expected to be seen in the data set. Multiplying this value further by $\hat{\epsilon}$ gives the prior number of errors we assume are in the data set. One caveat is that α must be greater than 0 to have a well-defined Beta distribution. Thus, if the estimated error rate is exactly 0, then we set $\hat{\epsilon}$ to a small constant c_ϵ, representing an extremely low error rate (Line 9). In practice we set $c_\epsilon = 0.00001$.

Lines 11 - 14, loop through every cluster C_κ and every sketch point $r_{\kappa q}$ using Algorithm 3 to recursively split each sketch point set $\mathcal{X}_{\kappa q}$ into bins. The parameters $\hat{\epsilon}$ and α are used determine when to stop splitting and assign bin vector values, as we will explain below.

Algorithm 3: Algorithm RecursiveBisectionBins

Inputs : $\mathcal{Y} = \{y_1 \dots y_w\}$=set of markers, where each y_j is a vector of dimension n with entries $y_j(i) \in \{-1, 0, 1\}$, $\hat{\epsilon}$ = estimated error rate, α = parameter of Beta distribution

Output: bins = set of sets of markers belonging to the same bin vector

bins= $\emptyset$; $\mathcal{Z} = \emptyset$; $M = |\mathcal{Y}|$;
$\beta = \frac{\alpha}{\hat{\epsilon}} - \alpha$; $\sigma = \sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}}$;
maps = getMAPs($\mathcal{Y}$);
maxMap = $\max_{i=1,\dots,n}$ map(i);
if *maxMap* $> \epsilon + 2\sigma$ **then**
    maxMapInd = i, *s.t.* map(i) = maxMap;
    bina = $\emptyset$; binb = $\emptyset$;
    for $j = 1, \dots M$ **do**
        if $y_j(i) == 1$ **then**
            bina = bina $\cup\{y_j\}$;
        else if $y_j(i) == -1$ **then**
            binb = binb $\cup\{y_j\}$
        else
            $\mathcal{Z} = \mathcal{Z} \cup \{y_j\}$;
    b_a =getBinGenotypes(bina); b_b =getBinGenotypes(binb);
    for $y_j \in \mathcal{Z}$ **do**
        if $p(y_j \in bina) < p(y_j \in binb)$ **then**
            Move y_j from $\mathcal{Z}$ to binb
        else if $p(y_j \in binb) < p(y_j \in bina)$ **then**
            Move y_j from $\mathcal{Z}$ to bina
        else
            Move y_j from $\mathcal{Z}$ randomly to bina or binb
    b_a =getBinGenotypes(bina); b_b =getBinGenotypes(binb);
    for $y_j \in bina$ **do**
        if $p(y_j \in bina) < p(y_j \in binb)$ **then**
            Move y_j from bina to binb
    for $y_j \in binb$ **do**
        if $p(y_j \in binb) < p(y_j \in bina)$ **then**
            Move y_j from binb to bina
    if $bina = \emptyset$ **then**
        return binb;
    else if $binb = \emptyset$ **then**
        return bina;
    else
        return(RecursiveBisection(bina,$\hat{\epsilon}, \alpha$) $\cup$RecursiveBisection(binb, $\hat{\epsilon}, \alpha$));
else
    return $\mathcal{Y}$;

3.3.2 Recursive Bisection with Error Correction

Algorithm 3, details the `RecursiveBisectionBins` function, which recursively bisects an input set of markers $\mathcal{Y}$, attempting to keep together markers belonging to the same bin. Recursion stops when the a posteriori estimate of the error rate for each individual in the input set of markers is close enough to the estimated rate $\hat{\epsilon}$. The output is a set of bins that markers in $\mathcal{Y}$ map to.

Recall that the input set $\mathcal{Y}$ will be one of the sketch point sets $\mathcal{X}_{\kappa q}$ from Algorithm 2, and thus we assume there are few bins represented by the markers in $\mathcal{Y}$ relative to the number of bins represented by the entire data set $\mathcal{X}$. The estimated error rate $\hat{\epsilon}$ and the parameter α are also given, which are used to set the prior Beta distribution over ϵ.

The `getMAPs` function (Line 3) finds maximum a posterior (MAP) estimate of the error rate for each individual i, based on a prior Beta distribution. The prior Beta distribution is set to have mean $\hat{\epsilon}$ using the parameter α. Thus, β must be set to $\frac{\alpha}{\hat{\epsilon}} - \alpha$ (Line 2). The variance of the Beta distribution is then additionally computed. Given that the posterior probability distribution over ϵ is proportional to the likelihood of the data $\mathcal{Y}$ times the prior distribution over ϵ, $p(\epsilon|y_1, \ldots, y_\eta) \propto p(y_1, \ldots, y_\eta|\epsilon)p(\epsilon)$, if all the markers $y_j \in \mathcal{Y}$ were generated from the same bin b, then: $p(\epsilon|y_1, \ldots, y_\eta) \propto (\epsilon)^o(1-\epsilon)^\eta \epsilon^{\alpha-1}(1-\epsilon)^{\beta-1} = (\epsilon)^{o+\alpha-1}(1-\epsilon)^{\eta+\beta-1}$, where o represents the number of entries $y_j(i)$ that are opposite from the true bin value $b(i)$, and η is total number of non-missing entries for individual i (the number of entries $y_j(i) \neq 0$). To compute a MAP estimate $\bar{\epsilon}$ for individual i, we set the derivative of $p(\epsilon|y_1, \ldots, y_{nm})$ to 0 to get: $\text{map}(i) = \frac{\alpha+o-1}{\beta+\alpha+\eta-2} = \bar{\epsilon}$. The higher the ratio $\frac{o}{\eta}$, the less likely that the set of markers $y_1, \ldots y_\eta$ have been drawn from the same bin. Additionally, more non-missing data (higher η) gives more weight to the MLE estimate of the error rate over the prior estimate, helping us leverage additional data for better estimates. In Line 4, we select

the maximum MAP estimates of the error rate.

If the maximum MAP estimate of the error rate is not within two standard deviations of the mean (Line 5), then it can be reasonably assumed that all the markers in $\mathcal{Y}$ could not have come from the same bin. Since the normal distribution with mean and variance equal to that of the Beta distribution is a good approximation to the Beta distribution near the tails, a MAP larger than this threshold indicates that the most likely error estimate for at least one individual is larger than approximately 97% of the error rates that we can expect to see based on our prior distribution. In other words, the data $\mathcal{Y}$ for individual i is inconsistent with our prior assumption. Thus, we assume that $\mathcal{Y}$ contains more than one bin, and we use the values of markers $y \in \mathcal{Y}$ at individual i to split $\mathcal{Y}$ into two subsets, called bina and binb. For a given marker y, if $y(i) = 1$, we assign y to bina, if $y(i) = -1$ we assign it to binb, and if $y(i) = 0$ (missing) it is temporarily assigned to a set $\mathcal{Z}$ (Lines 6 - 14).

To assign the markers y in $\mathcal{Z}$, we temporarily assume that bina and binb represent true bins, and calculate the probability that y belongs to either bin. To do so, the `getBinGenotypes` function (Algorithm 4) first assigns a bin vector b_a to bina and a bin vector b_b to binb. The `getBinGenotypes` function loops over all individual entries i and assigns a genotype to the ith entry of the bin vector if the MAP estimate of the error rate at the ith entry is within 2 standard deviations of the mean of the Beta distribution. If so, `getBinGenotypes` assigns the entry to the maximally-occuring genotype at that individual entry. If not, then `getBinGenotypes` assigns a 0 to this entry, effectively stating that there is not enough evidence to assign this entry a value.

Then, using the assumption that errors are binomially distributed the probability

that y belongs to bina becomes:

$$p(y \in \text{bina}) = \prod_{i=1}^{n} (1-\epsilon)^{b_a(i)=y(i),y(i)\neq 0} \, \epsilon^{b(i)\neq y(i),y(i)\neq 0} \tag{3.1}$$

and analogously for binb. The `for` loop on line 16 assigns each y in $\mathcal{Z}$ to the side of the partition which yields a greater probability, breaking ties randomly.

After the Zero-Assignment phase, the initial input set $\mathcal{Y}$ has been partitioned into two subsets that are more consistent with the error rate for one individual. However, only one bit of information per marker was used to assign it to one of these subsets. An error in this bit would thus cause the marker to be assigned to the wrong set, and the error would "pass down" the recursion. Therefore, after bisecting the markers we add an error correction phase (Line 24).

The Error-correction phase calculates, for each marker y, its probability of belonging to each side of the bisection. We again use the `getBinGenotype` function to assign a bin vector to each side of the partition and then calculate the probabilities according to equation 3.1. If for any marker $y_j \in$bina it is found that $p(y_j \in \text{bina}) < p(y_j \in \text{binb})$, y_j is moved from bina to binb (Lines 25-26), and vice-versa (Lines 28 - 29).

After the error correction phase, if either bina or binb is empty (Lines 30 and 32), the bin vector corresponding to the opposite bin is returned. Here we assume that we attempted to split $\mathcal{Y}$ based on the most informative individual, in terms of maximum MAP estimate of the error rate, but it is more likely that the set of markers represents a bin than that it is a combination of bins. Therefore, we return the bin corresponding to the entire set of markers. Otherwise, the code recursively partitions bina and binb using the same algorithm (Line 35), until each column in the input set of markers is consistent with the error rate. In this case (Line 36), we return the input set as a bin. In other words, all the markers are assumed to have come from the same bin.

Algorithm 4: Algorithm getBinGenotypes

Inputs : $\mathcal{X} = \{x_1 \dots x_M\}$=set of markers,
where each x_j is a vector with each entry $x_j(i) \in \{-1, 0, 1\}$, and each x_j has n individual entries,
ϵ = estimated error rate,
α = parameter of Beta distribution

Output: b = the genotypes of the bin assumed to have generated $\mathcal{X}$, with $b(i)$ = genotype of individual i

```
1  o =number of positive genotypes for each individual i;
2  η =number of non-missing genotypes for each individual i;
3  maps = getMAPs(X);
4  β = α/ε − α; σ = sqrt(αβ / ((α+β)²(α+β+1)));
5  mapthreshold = ε + 2σ;
6  for i = 1...n do
7      maxgenotype = max(o(i), η(i) − o(i));
8      if (maps(i) < mapthreshold) ∧ (maxgenotype = o(i)) then
9          b(i) = 1;
10     else if (maps(i) < mapthreshold) ∧ (maxgenotype = η(i) − o(i)) then
11         b(i) = -1;
12     else
13         b(i) = 0;
```

3.3.3 Run Time Analysis

The BubbleCluster algorithm runs in $O(|\mathcal{H}| \log |\mathcal{H}| + M\bar{R})$, where M is the data set size, $\bar{R}$ the number of sketch points found and $|\mathcal{H}|$ the number of high-quality markers used in the sketch point-building phase[68]. We note that $|\mathcal{H}|$ is typically much smaller than M, and in our experiments we this was indeed the case. The number of sketch points $\bar{R}$ is bounded by kn, the number of individuals n times the number of linkage groups k, and is thus much smaller than both $|\mathcal{H}|$ and M. The running time of the error estimate in line 7 of Algorithm 2 is linear in M times the number of sketch points, $\bar{R}$. The `getBinGenotypes` function (Algorithm 4) runs in time that is linear in the input set size.

The running time of recursive bisection binning is complex to analyze, because the

depth of recursion depends on the number of times a MAP estimate for a particular individual will be greater $\hat{\epsilon} + 2\sigma$. We note, however, that if we rely on our assumption that the markers in a sketch point set $\mathcal{X}_{\kappa q}$ are very close together on the genome, then for a majority of individuals, the MAP estimate will be consistent with the error rate. Therefore, only a few splits are required before all markers in a set $\mathcal{Y}$ agree, up to $\hat{\epsilon} + 2\sigma$, in individual entries. Thus if we are successful in producing a coarse-grained clustering of the markers within each cluster into sketch point sets $\mathcal{X}_{\kappa q}$, such that each sketch point set contains at most n/λ bins, with $\lambda > 1$, then the depth of recursion is bounded by n/λ with high probability. In this case, recursive binning takes time proportional to $\sum_i \sum_j |\mathcal{X}_{\kappa q}| n/\lambda = \frac{nM}{\lambda}$, and the entire algorithm requires a runtime of $O(|\mathcal{H}| \log |\mathcal{H}| + 2M\bar{R} + \frac{nM}{\lambda})$. Since n, $\bar{R}$, and $|\mathcal{H}|$ are much smaller than M, our algorithm will run in time that is linear in the input size, *assuming we obtain a good coarse-grained clustering*. Experimental results in Section 3.4 validate that this is empirically correct.

3.4 Experimental Results

We use several metrics to evaluate the quality of our binning algorithm using both real and simulated data. Our code is written in C++, single threaded, and available at http://gauss.cs.ucsb.edu/home/index.php/code. We ran experiments on a single core of one 12-core compute node of NERSC's Carver system, with Intel Xeon X5650 processors running at 2.67GHz, and 10GB of memory. In reporting our experimental results we will denote by f_i the ith bin vector found by our algorithm and the set of found bins $\mathcal{F}$. For experiments on simulated data, we will use the notation b_l to denote the lth golden standard bin vector.

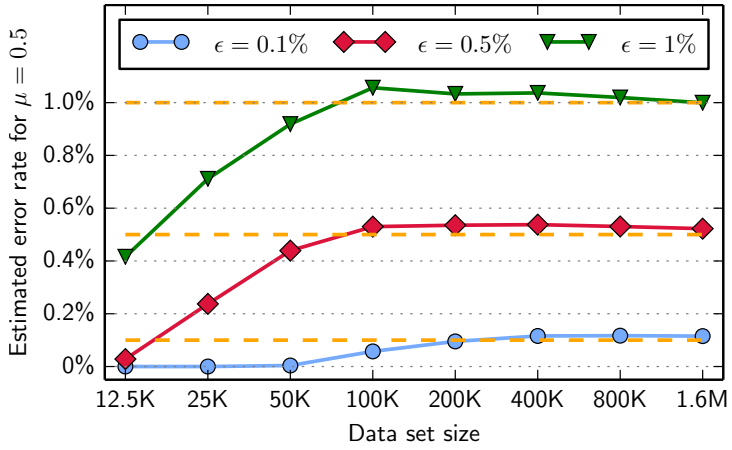

Figure 3.4: Estimated error rates $\hat{\epsilon}$ for missing rate $\mu = 0.5$ with increasing data set sizes. As the data set size grows, $\hat{\epsilon}$ approaches the true rate ϵ for each case tested: $\epsilon = 0.1\%$, 0.5%, and 1%.

3.4.1 Simulated Data Sets

To evaluate the quality of our binning algorithm, we use simulated data sets ranging in size from 12.5K to 1.6M markers, with realistic and challenging missing rates [67, 65, 69, 62, 70] $\mu = 0.15, 0.35, 0.50$ and error rates $\epsilon = 0.1\%, 0.5\%, 1\%$. All data sets of size less than 1.6M are randomly chosen subsets of the 1.6M marker set for a fixed (μ, ϵ) pair. Experiments on real genomic data for barley and wheat are presented in Section 3.4.6.

To approximate realistic genetic map sizes, and for a thorough comparison between varying missing rates and error rates, we set the number of chromsomes k to 10 and the number of individuals in the mapping population n to 100 in all experiments. For each chromosome, we simulated n bin vectors in the following manner. An initial bin vector b_1 was generated by randomly assigning $b_1(i)$ to -1 or +1 with equal probability for all n individuals. Next, for $j \in \{2, \ldots, n\}$ b_j was generated by randomly choosing an individual $i \in \{1, \ldots, n\}$, and flipping the sign of the ith individual in b_{j-1}. Thus, each bin vector $b_j, j \in \{2, \ldots n-1\}$ differs in exactly one individual entry from b_{j-1} and from b_{j+1} (and b_1 and b_n differ in exactly one individual from b_2 and b_{n-1}, respectively). Note that this procedure corresponds to simulating one recombination per chromosome per individual, a

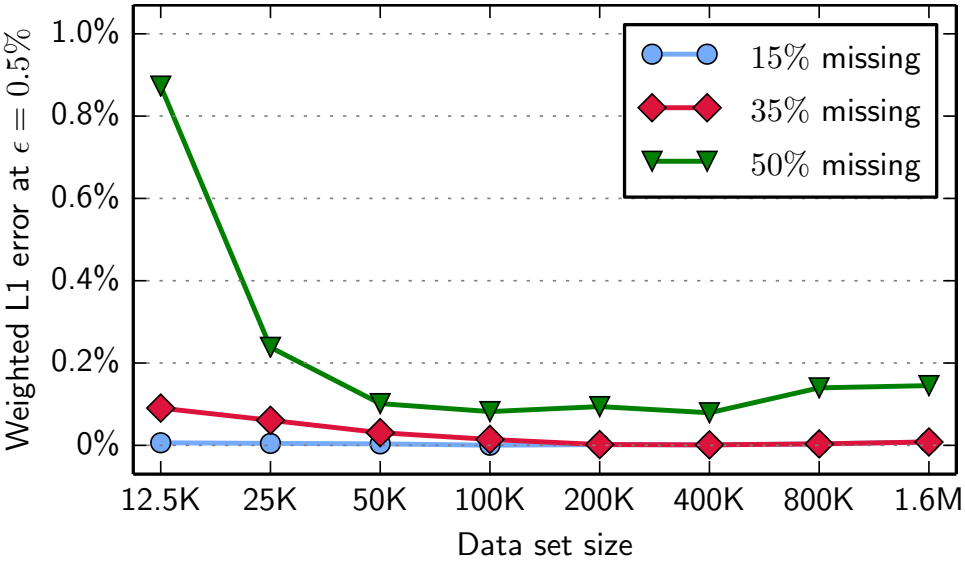

Figure 3.5: Weighted L_1 error for fixed error rate $\epsilon = 0.5\%$ and missing rates $\mu = 0.15, 0.35, 0.50$. In all cases, average L_1 distance from a found bin vector to the golden standard was less than 1.

common and realistic feature of true mapping populations [39, 5, 49, 50, 66]. Importantly, this also means that the fundamental resolution of our data was 1000 in all simulated experiments.

Once the golden standard bin set has been created, we simulate the markers. First, a bin vector b_q is chosen uniformly at random. Then, as long as the number of simulated markers is less than M, a marker is generated from the qth bin. A marker x_j is generated from bin vector b_q by setting the entry $x_j(i)$ is to $b_q(i)$ with probability $1 - \epsilon$, and to $-1 * b_q(i)$ with probability ϵ. The entry $x_j(i)$ is set to 0 with independent probability μ. Thus, as we increase the data set sample size M, more markers are sampled from each bin at the same rate in expectation.

3.4.2 Error Estimation & Reduction Accuracy

To evaluate the quality of our algorithm, we quantify how well the true error rate ϵ compares with our estimate $\hat{\epsilon}$, and report three metrics to evaluate how accurately our

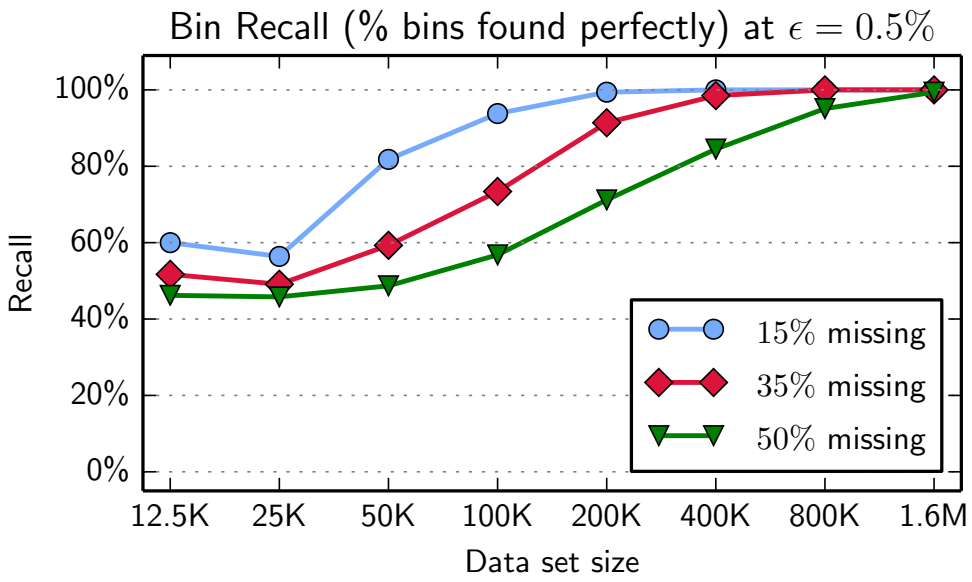

Figure 3.6: Bin recall for a fixed error rate $\epsilon = 0.5\%$ with missing rates of $\mu = 15\%$, 35%, 50%. As the data set grows, more bins are recovered perfectly, while lower missing rates allow our algorithm to recover the bins with less data.

algorithm reduces a set of M markers to a set of $|\mathcal{B}|$ bins. *Accuracy* is defined as follows: to each bin vector f_j discovered by our algorithm, we assign one golden standard bin vector b_l based on closest L_1 distance, breaking ties randomly. Then, we report accuracy as the average over all found bin vectors f_j of the number of matching individual entries where $f_j(i) = b_l(i)$, divided by the total number of non-missing entries for which $f_j(i) \neq 0$. In other words,

$$\text{accuracy} = \sum_{j=1\ldots|\mathcal{F}|} \frac{1}{\eta_j} \sum_{i=1..n} (f_j(i) == b_l(i))$$

where for each f_j, b_l is the closest golden standard bin in terms of L_1 distance and η_j is the number of non-missing entries in f_j. Thus, if accuracy is 100%, then *every* found bin vector f_j corresponds exactly to some golden standard bin vector b_l.

The second metric used to evaluate accuracy is referred to as *weighted* L_1 *error*. This is the average L_1 distance from a found vector f_j to its closest-matching (in terms of L_1 distance) golden standard vector b_l, weighted by the size of the set of markers assigned

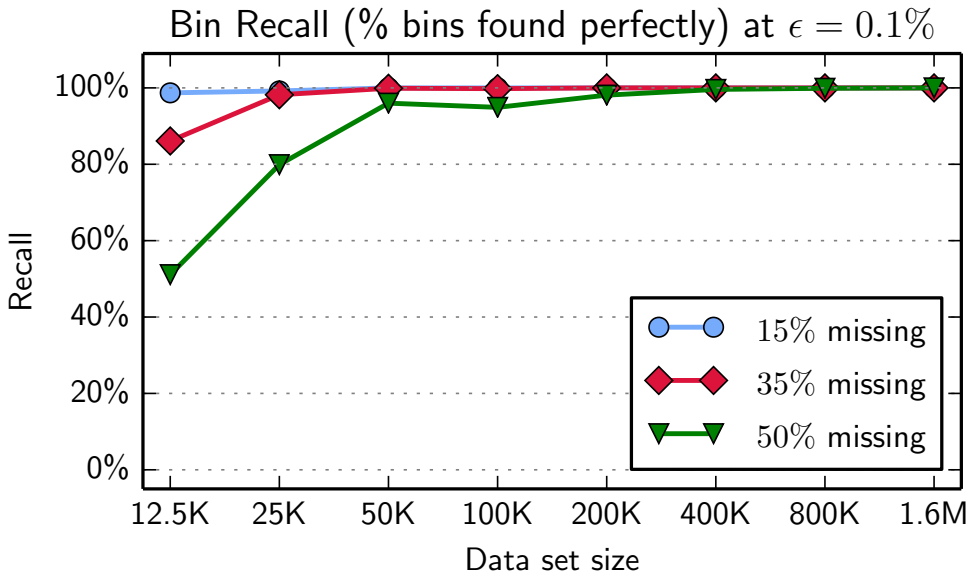

Figure 3.7: Bin recall for a fixed error rate $\epsilon = 0.1\%$ with missing rates of $\mu = 15\%, 35\%, 50\%$. As the data set grows, more bins are recovered perfectly, while lower missing rates allow our algorithm to recover the bins with less data. With the low (but realistic) error rate of $\epsilon = 0.1\%$, we perfectly recover all true bin vectors with less required data than when $\epsilon = 0.5\%$ error.

to bin vector f_j. Note that an L_1 distance of 2 indicates one incorrect entry in f_j, a distance of 4 indicates 2 incorrect entries, and so on. For example, the L_1 distance between $f_j = (1, 1, 1, -1)$ and $g_l = (1, 1, 1, 1)$ is 2, because $|f_j(4) - g_l(4)| = 2$ and $|f_j(i) - g_l(i)| = 0$ for $i \in \{1, 2, 3\}$. Formally, we define the weighted L_1 error as:

$$\text{L1 error} = \frac{1}{m} \sum_{j=1\ldots|\mathcal{F}|} |\text{found bin}_j| \sum_{i=1,\ldots,n} |f_j(i) - g_l(i)|$$

where m is the data set size, and $|\text{found bin}_j|$ is the number of markers assigned to bin vector f_j. Therefore, larger bins contribute more to the calculation of average L_1 error.

Finally, we report a third metric that we name *recall*, defined as the percentage of

true bin vectors that were found perfectly:

$$\text{recall} = \frac{\sum_{l=1\ldots|\mathcal{B}|}(f_j = b_l)}{|\mathcal{B}|}$$

3.4.3 Simulated Data Experiments

Figure 3.4, plots the error rate $\hat{\epsilon}$ for increasing data set size and three different ϵ values, with μ fixed at 50%. We chose ϵ to reflect true genotyping errors in genetic marker data, which rarely rise above 0.5%[70]. Observe that even with 50% missing data in the input, with enough data our estimate accurately approaches the true error rate for each ϵ tested — 0.1%, 0.5%, and 1%.

Next, we report the accuracy and recall for increasing data set size and varying error rates, with μ fixed at 50%, shown in Table 3.1. Observe that the trend agrees with our intuition — more data is required to correctly recover the true bin vectors as the error rate grows. However, notice that the found bin vectors are extremely close to the true bin vectors; the accuracy is above 95% in all cases. The default constant c_ϵ was used to set $\hat{\epsilon}$ only for the 12.5K and 25K cases for $\epsilon = 0.1\%$. We achieve accuracy above 99% in all cases when the data set size is 100K or greater, meaning that on average, the bin vectors we find agree in 99 or more positions out of 100 with the golden standard.

In our next set of results, Figure 3.5, we present the weighted L_1 error for increasing data set size, with variation in the missing rate μ for a fixed ϵ at 0.5%. Observe that with a low missing rate of 15%, the errors made in bin vector assignments values are low even for small data set sizes, and remain close to 0 as the data size increases — confirming that increasing the amount of data for a low missing rate does not hurt binning accuracy. For a moderate missing rate of 35%, we see that the weighted L_1 error starts near 0.1, indicating that on average, large bins correspond to bin vectors that are within 0.1 L_1

distance of true bin vectors. Of course, the L_1 distance is always a positive integer in this case. Therefore, we can interpret this error as the effect of the extra, small-sized bins that are found in addition to the golden standard vectors. An important point demonstrated by the weighted L_1 error is that these extra bin vectors are still close to the underlying true bin vectors. In other words, few bin vector values are assigned incorrectly and those erroneous bins are themselves extremely close in L_1 distance to true bin vectors.

The error for the 35% missing rate quickly decreases toward 0, showing that approximately 200,000 markers are sufficient to find the 1,000 bin vectors that generated the data almost perfectly. Finally, we observe that the 50% missing rate trend line actually *increases* at some increments of data size. However, as we will discuss next, the loss in accuracy is offset by the gain in recall.

Figure 3.7(left) shows the bin vector recall for a fixed $\epsilon = 0.5\%$ and varying missing rates μ. Note the similar trend to the weighted L_1 error results — as the missing rate increases, more data is required to perfectly recover all the true bin vectors. However, more interestingly, even the low missing rate of 15% requires 200K markers to find all the golden standard bin vectors. With 50% missing values, it is not until we process the

Data Size	Error Rate (ϵ)					
	0.1%		0.5%		1%	
	Accur	Recall	Accur	Recall	% Accur	Recall
12.5K	95.87	51.10	98.76	46.2	99.3	27.3
25K	96.13	79.90	99.81	45.8	99.67	33.1
50K	98.44	96.0	99.93	48.7	99.85	39.2
100K	99.88	94.9	99.93	56.8	99.88	47.9
200K	99.87	98.1	99.92	71.2	99.89	59.7
400K	99.87	99.6	99.94	84.5	99.93	70.7
800K	99.80	99.9	99.89	95.1	99.91	85.5
1.6M	99.76	100	99.87	99.4	99.87	95.9

Table 3.1: Accuracy and recall (in %) for fixed $\mu = 50\%$ and increasing error rate (ϵ). In most cases, accuracy is above 98%. Higher error rates require more data to recover all true bin vectors.

1.6M marker data set that perfect recall is achieved. These results highlight the necessity for large data sets when missing data rates are high, even with relatively low error rates. An exciting implication of this result is that with enough inexpensive, low-coverage DNA sequencing, which produces much higher missing rates than more expensive, repetitive high-depth sequences [70], we can effectively recover the fundamental information for constructing the genetic map. To show the effect of the error rate on bin recall, we also present a recall plot for ϵ fixed at 0.1% in Figure 3.7(right). The lower error rate allows our algorithm to recover true bin vectors at much smaller scales than witnessed in Figure 3.7(left).

3.4.4 Computational Efficiency

We now examine the computational efficiency of our algorithm. Runtimes for increasing data set size, with μ fixed at 50%, are shown using $\epsilon = 0.1\%$, 0.5%, and 1% in Figure 3.4.4. Even up to 1.6 million markers, we observe linear running time of our algorithm. Importantly, this supports the idea that a coarse initial clustering sufficiently partitions the data into groups that represent markers close together on the genome, and that can be quickly further subdivided into bins. Our experiments across missing rates (not shown) also resulted in linear scaling for all tested cases.

3.4.5 Variable Bin Sizes

In addition to handling bins of approximately uniform size, our algorithm is designed to perform well even when, as is sometimes the case in real data, a few large bin sizes dominate the data set. To test this claim, we generated simulated data in the same manner we described in Section 2.4.1, with one modification. For each simulated linkage group, instead of creating n bins of approximately equal size, we randomly select 3 bin

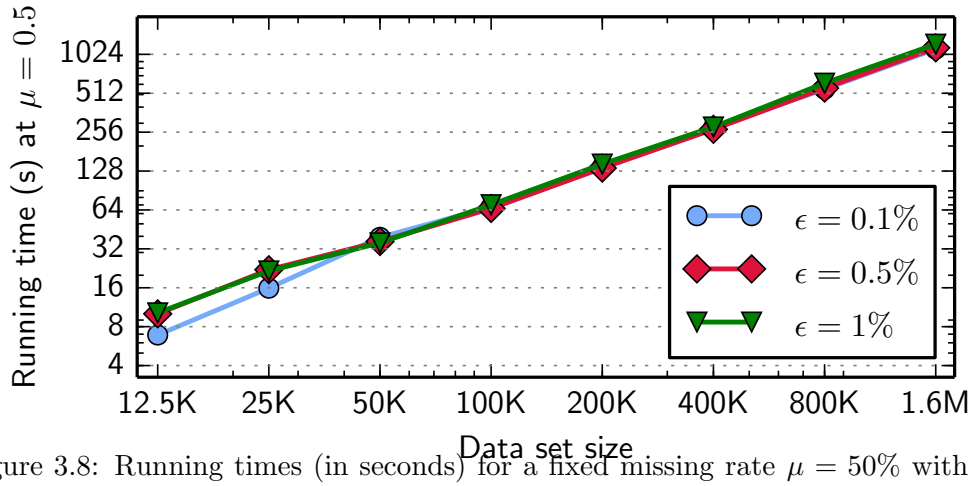

Figure 3.8: Running times (in seconds) for a fixed missing rate $\mu = 50\%$ with error rates $\epsilon = 0.1\%, 0.5\%, 1\%$. Both axes are on a logarithmic scale, as the data set size grows, running time grows linearly.

vectors, and generate 20x more markers from these bin vectors than the others. Thus, the data is dominated by three large bins per linkage group. The number of linkage groups k remains fixed at 10 and the population size n remains fixed at 100.

The results for variable bin sizes with ϵ fixed at 0.5%, and μ fixed at 50%, are shown in Table 3.2. We report accuracy, recall, and running time for increasing data set size. The quality of the bin vectors found by our algorithm remains high — in all cases, over

Variable Bin Size Experiments			
Data size	Accuracy (%)	Recall (%)	Running Time(s)
12.5K	99.79	33.3	39.2
25K	99.76	37.7	33.8
50K	99.87	49.2	48.2
100K	99.97	56.8	83.0
200K	99.98	71.5	150.8
400K	99.98	90.2	310.6
800K	99.98	98.8	567.1
1.6 M	99.98	100	1180.0

Table 3.2: Results for variable bin sizes, with a fixed $\mu = 35\%$ and $\epsilon = 0.5\%$. Runtime scales linearly, and accuracy increases, with larger data sizes. Because a few large bins dominate the input data, a large data set size is required to discover all true bin vectors.

99% of the vector entries were assigned correctly. However, we recover fewer bins than in the corresponding uniform bin size case (Figure 3.7) per data set size. Here again we see the advantage of using large-scale data: whereas smaller data sets do not hold enough information to identify smaller bins, our algorithm utilizes large data sets to correctly recover both small and large bins, without introducing errors to the binning accuracy. Running time remains linear with respect to data set size.

3.4.6 Real Data Experiments

We include results on two real data sets in Table 3.3. The barley data [67] contains $65,357$ genetic markers from a population of 90 individuals and 7 linkage groups, with 20% missing data. The wheat data consists of 1.7 million genetic markers, from a population of 88 individuals and 21 linkage groups, with missing rate 39% [65]. We do not have true bin vectors available to test for accuracy and recall on these data sets, and thus report the estimated error rate, the running time, and number of bins found by our algorithm.

For both barley and wheat, the estimated error rate is close to 0.1%, a realistic estimate given the sequencing methods used to generate the data [67, 65, 69, 62, 70]. The time to output the bin vectors was less than a minute for barley and only 23.5 minutes for the large, complex wheat genome. Note that our wheat bin solution is within 5% (1410 vs. 1335) of previously published analysis [65]. Overall this demonstrates that our methodology enables the capability for the gene mapping community to effectively and quickly utilize multi-million marker data sets.

Results for Real Data				
Data	Size	Estimated ϵ	Time(s)	Bins Found
Barley	65K	0.102%	45.9	467
Wheat	1.7M	0.064%	2728.0	1410

Table 3.3: Results for barley and wheat data. Our algorithm quickly reduces the data set size and estimates an error rate consistent with sequencing technologies used to produce the two data sets.

3.4.7 Upstream Analysis

Finally, we provide an initial analysis of the effect of genetic marker data reduction on the quality of the genetic map produced by taking our bin vectors as input. We use MSTMap [39], a popular genetic mapping tool, to produce a genetic map from the bin vectors of our algorithm. We use a set of 50K and 800K simulated markers with 35% missing data, 0.5% error rate, and uniform bin size. To assess the quality of this map, we compare it to the map produced by MSTMap when given the true bin vectors as input.

Table 3.4 displays two points of comparison between the MSTMap output on true versus found bins. First, we report Spearman's correlation coefficient, ρ, between the true bins as they are ordered when processed by MSTMap, and their order using MSTMap on our set of found bins. In other words, we use MSTMap to order and report genetic distances for both the gold standard bin set and the set found for our algorithm (inputting the bin vectors corresponding to only one linkage group at a time). Then, for each bin f_j that our algorithm found corresponding exactly to a true bin g_l, i.e. $f_j = g_l$, we compare its order in the found set to its order in the gold standard set. Recall that our algorithm found all true bins perfectly in the 800K case, but not in the 50K case. For both data sizes, our algorithm also produced a few spurious bins in each linkage group, with low L_1 distance to true bins (Figures 3.5 and 3.7). Table 3.4 shows that for all linkage groups, $|\rho|$ was greater than 0.999, with higher quality for the larger 800K data set. This result confirms our intuition that the spurious bin vectors produced by our algorithm do not

MSTMap Ordering: True vs. Found Bins		
Data Size	50K	800K
Spearman's $\|\rho\|$	[0.9993 – 1]	[0.9997 – 1]
Relative Map Increase (% cM)	[1.1 – 19.4]	[4.0 – 12.1]

Table 3.4: Comparison of genetic maps produced by MSTMap using true bins, vs. found bins using 35% missing data and 0.5% error rate, showing the range of solutions for all 10 linkage groups. Note that although our binning approach increases the map size, the mapping relative to the true bins is almost perfectly preserved.

introduce large errors in the final map product.

The effect of spurious bins is apparent in the relative map sizes of the found vs. true bins, shown in Table 3.4. MSTMap outputs a map size in terms of genetic distance, or *centimorgans* (cM). We compare the size of the map produced by feeding MSTMap the bins found by our algorithm, to the size of the MSTmap produced using only golden standard bins. Observe that in all cases, the map size increases, due to extra bin vectors within the map. However, the true bins remain almost perfectly ordered. This is a promising initial result — we have quickly reduced a data set with a size that is unmanageable by MSTMap to a more accurate and complete set that almost perfectly preserves the mapping quality of the underlying true bin set. Future work will explore an optimized solution to the ordering and mapping problem based on our reduced bin input.

3.5 Related Work

Several existing theoretical works provide bounds on the data set size required to solve the related *matrix completion*, *co-clustering*, and *dimension reduction* problems. In genetic mapping research, although some notion of bins exists in several tools, we are unaware of any method that applies data reduction to the characteristically noisy and

incomplete genetic marker data to efficiently and accurately discover the fundamental set of vectors making up the genetic map.

In the matrix completion problem, a matrix X is given as input with missing entries, as well as potentially noisy entries, with the goal of filling these entries correctly [71, 72]. Candes et al. show [71] that a sample of $m \geq CM^{1.2}r \log M$, where M is the data set size and r is the rank of the data matrix, is sufficient to perfectly recover the full matrix in most cases. Differently from our problem, the algorithm does not address errors in data entries. Xu et al. do address the matrix completion and noise reduction problem [72], for the specific case of co-clustering binary matrices, that is, clustering the rows and columns of an input binary matrix simultaneously. The most relevant result [72] to our work is that if the binary input matrix is block-constant, all entries can be recovered exactly with a sample of size $O(Mr^2)$, where r is the matrix rank. This interesting result cannot be applied directly to our problem as we do not assume a block-constant structure to our matrix.

Dimensionality reduction is a broad field of research with numerous applications. The goal is to reduce the input data to some lower-dimensional space which more accurately captures the structure of the data [73, 74, 75, 76]. Because we assume that all entries in our data matrix are generated by a small, finite set of bin vectors, our problem may be restated as reducing the set of markers to the n-dimensional hypercube of bin vectors. Many methods have been proposed to reduce the dimensionality of input data. We attempted to apply locally distance-preserving methods such as maximum variance unfolding [74] and locally linear embedding [76] to our data, using the LOD score as a similarity measure. However, these methods failed to map markers from the same bin to the same location in the lower-dimensional space, and they are computationally expensive. Bailey [73] addresses the problem of principal component analysis (PCA), which can be used to project input data to lower dimensions, of a noisy input matrix with missing

data. However, data errors are assumed to be small changes to matrix entries, unlike the errors in our binary input data, which cause an entry to be changed to the value completely opposite its true value. Furthermore, the desired dimensionality of our data reduction is unknown a priori, and PCA requires the number of principal components k to be provided as input.

Our algorithm is similar in spirit to the unsupervised decision tree approach of Karakos et al. [77] and to the coarse-to-fine clustering approach of the BIRCH [60] and CURE [61] algorithms. The recursive construction of *Integrated Sensing and Processing Decision Trees* (ISPDTs) is used [77] to perform unsupervised classification. The initial data set is recursively split into subsets with the goal of discovering the underlying classes that generated the data with high probability, a similar problem to our binning objective. Expensive and greedy heuristics are applied at each step to either (a) maximize mutual information between the unknown class and the path from root to leaf in the ISPDT, or (b) minimize the probability of misclassification. Initial results on multispectral imaging data seem promising, but no running time analysis is provided, and it is unclear how the method performs on large-scale data sets.

BIRCH and CURE fall under the category of hierarchical clustering algorithms, which attempt to first reduce data input size into a coarse clustering, then refine the clustering within each coarse partition. However, the efficiency and accuracy of these algorithms relies on the assumption that points lie in a Euclidean d-dimensional space, and perfect data, which contains no errors or missing entries. These algorithms are also superlinear in the input data set size.

Existing genetic mapping tools include OneMap [51], MSTMap [39], and JoinMap [45]. Many of these tools provide a notion of bins as uniquely identifiable locations on the genetic map. Of these tools, MSTMap has gained popularity in recent years due to its relatively low computational complexity in comparison with other genetic mapping

software. MSTMap also applies a simple heuristic algorithm to attempt to reduce the data into sets of *co-segregating* markers, or markers whose Hamming distance is 0, before beginning the map building process. The goal is of this algorithm is not to reduce the scale of the data to its fundamental resolution nor to reduce errors and fill in missing data. Thus it would be inaccurate to directly compare our bin output to MSTMap's sets of co-segregating markers. However, MSTMap requires quadratic time in the input data size, and thus is prohibitively slow on large data sets. Wu et al. also note that the quality of the map produced by MSTMap drops significantly with high error and missing rates [39]. For example, on a 25K simulated data set with 35% missing rate and 0.5% error, MSTMap runs in 2577.5 seconds to produce a map, compared with only 4.0 seconds when given golden standard bins as input. Therefore, our algorithm enables these tools to effectively handle modern-day, large-scale noisy data sets produced by next generation sequencing technologies.

3.6 Discussion

We have introduced an algorithm for large-scale genetic marker reduction, and shown that it can quickly reduce the size of large-scale, noisy, and incomplete genetic marker data almost perfectly to a fundamental set of bin vectors that define unique locations on a genetic map. Even with challenging data sets with error rates as high as 1% (which are unlikely in real data), missing rates as large as 50%, and variable bin sizes, our algorithm correctly reproduced the vectors that correspond to the fundamental set size when given enough data. Furthermore, the reduction in data size enables traditional software tools, designed for the small-scale setting, to perform well on reduced large data sets.

Our experimental results can also serve to guide future genetic map building efforts. By simulating various missing and error rates, we have shown experimentally, for a typical

genetic map population size, how much data is required to correctly recover all bin vectors, given a fundamental set size. Our results suggest that although our algorithm works well on low missing and error rates, we can utilize large amounts of data with high missing and error rates to correctly recover bins. Notably, we demonstrate the near-linear efficiency of our algorithm on both simulated and real data, up to a 1.7M wheat data set.

Future research will extend our algorithmic work to the final phase of genetic mapping: ordering the bin vectors to produce a final map. Although existing tools such as MSTMap perform reasonably well given high-quality, small-scale data, we believe that the properties of the bin vectors found by our algorithm can be leveraged to aid genetic map construction for the massive data sizes of next-generation sequencing technologies.

Chapter 4

A New Similarity Score for Recommender Systems

Recommender system data presents unique challenges to the data mining, machine learning, and algorithms communities. The high missing data rate, in combination with the large scale and high dimensionality typical of recommender systems data, requires new tools and methods for efficient data analysis. Here, we address the challenge of evaluating similarity between users in a recommender system, where for each user only a small set of ratings is available. We present a new similarity score, that we call LiRa, based on a statistical model of user similarity for large-scale, discrete valued data with many missing values. We use an empirical evaluation on real data to show its effectiveness in finding similar users in user-based collaborative filtering. We also present an evaluation of several similarity scores' ability to detect clustered points in synthetic data sets, revealing fundamental properties of these scores that are important in their application to recommender system data. We show that LiRa is more effective at identifying similar users than traditional similarity scores in user-based collaborative filtering, such as the Pearson correlation coefficient. We argue that our approach has significant potential to improve both accuracy and scalability in collaborative filtering.

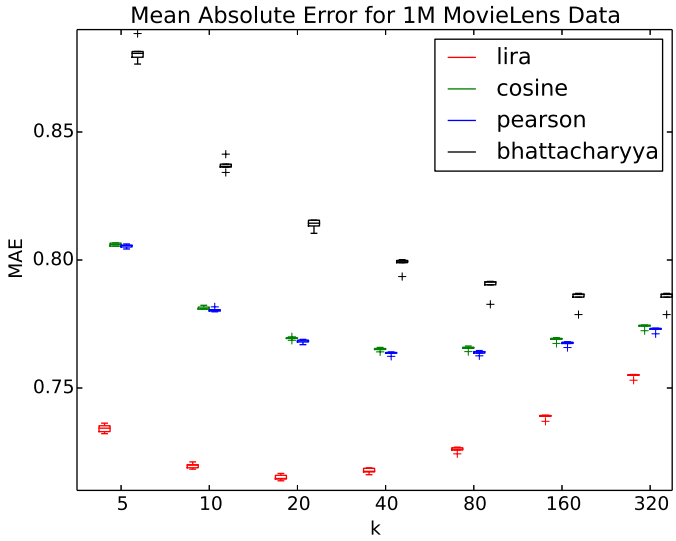

Figure 4.1: Prediction accuracy of the kNN method using several similarity scores on the 1M MovieLens dataset. The LiRa similarity attains the lowest MAE across tested values of k. The greater difference between LiRa and other scores at lower values of k indicate its better ability to find similar users.

4.1 LiRa Score: Motivation and Background

Our work stems from a well-known problem in collaborative filtering: RS data is often very sparse, meaning that in a system with m users and n items, the number of ratings observed is typically much less than the mn user-item pairs. Thus in approaches that seek to predict future ratings based on user-user or item-item similarity, it is important to consider how the sparsity of ratings affects the similarity score.

Popular choices of similarity scores for user-based collaborative filtering include the Pearson correlation coefficient and the cosine similarity, which are "commonly accepted as the best choice."[78] We review these traditional scores briefly to highlight the core issue that will be addressed with our new similarity score, presented in Section 4.2.

For two users u and v, let I_{uv} be the set of co-rated items, i.e. those items that were rated by both u and v. Let r_{ui} be the rating given to item i by user u. Then, the Pearson

correlation between users u and v, $PC(u,v)$, is defined as follows [78]:

$$PC(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}} \tag{4.1}$$

where $\bar{r}_u$ is the average rating given by user u to the items in I_{uv}, and similarly for $\bar{r}_v$. The Pearson correlation is a measure of linear correlation between user u and v's ratings, and takes on values between -1 and 1.

The Cosine Vector similarity between users u and v, or $CV(u,v)$, is a measure of the angle between the N-dimensional vectors defined by user u and v's ratings. More specifically, if we let $y_u \in R^N$ be the vector with $y_{ui} = r_{ui}$ for rated item i, and $y_{ui} = 0$ otherwise, then:

$$CV(u,v) = \frac{y_u^T y_v}{||y_u|| ||y_v||} = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2 \sum_{j \in I_v} r_{vj}^2}} \tag{4.2}$$

where I_u, I_v are the sets of items rated by u and v, respectively. The cosine of the angle between two vectors ranges from -1 to 1, with 1 indicating perfectly matching entries in both vectors. A CV similarity of 1, therefore, indicates perfectly matching entries. However, although a cosine of 0 indicates orthogonal vectors in an N-dimensional vector space, the cosine of two rating vectors will only be 0 if the there are no co-rated items in raw (unnormalized) data.

A major issue with both of these scores is their lack of consideration for missing data. Although the PC similarity, which is equivalently the sample Pearson correlation coefficient, is a consistent estimator of the population correlation coefficient for large sample sizes, the number of co-rated items between u and v, or $|I_{uv}|$, is often so small that the PC similarity is not reliable.

Similarly, we can think of the Cosine Vector similarity as the cosine of the angle

between two vectors that represent the projection of user ratings onto the space spanned by the $|I_u \cup I_v|$ dimensions in which data is observed, but the value of this angle in higher dimensions has is treated the same as its value in lower dimensions.

We conclude this explanation of the drawbacks of popular similarity scores used on RS data with some examples.

First, suppose we want to compute the similarity between two users represented by the following identical rating vectors, constructed by following the definition of vectors used by the CV similarity:

$$x_u = \begin{bmatrix} 1 & 1 & - & - & - & 2 \end{bmatrix}, x_v = \begin{bmatrix} 1 & 1 & - & - & - & 2 \end{bmatrix} \tag{4.3}$$

Both CV and Pearson will yield a score of 1 between u and v:

$$\begin{aligned} CV(x_u, x_v) &= \frac{x_u^T x_v}{||x_u||||x_v||} \\ &= \frac{5}{\sqrt{5}\sqrt{5}} = 1 \end{aligned}$$

$$\begin{aligned} PC(u,v) &= \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}} \\ &= \frac{\sum_{i \in \{1,2,6\}} (r_{ui} - \frac{4}{3})(r_{vi} - \frac{4}{3})}{\sqrt{\sum_{i \in \{1,2,6\}} (r_{ui} - \frac{4}{3})^2} \sqrt{\sum_{i \in \{1,2,6\}} (r_{ui} - \frac{4}{3})^2}} \\ &= \frac{(-\frac{1}{3})^2 + (-\frac{1}{3})^2 + (\frac{2}{3})^2}{\sqrt{\left((-\frac{1}{3})^2 + (-\frac{1}{3})^2 + (\frac{2}{3})^2\right)} \sqrt{\left((-\frac{1}{3})^2 + (-\frac{1}{3})^2 + (\frac{2}{3})^2\right)}} = 1 \end{aligned}$$

If we increase the amount of data observed, leaving the vectors identical:

$$x_w = \begin{bmatrix} 1 & 1 & 5 & 4 & 4 & 2 \end{bmatrix} x_z = \begin{bmatrix} 1 & 1 & 5 & 4 & 4 & 2 \end{bmatrix} \tag{4.4}$$

the CV and Pearson similarities will remain the same:

$$CV(w,z) = \frac{x_u^T x_v}{||x_u||||x_v||}$$

$$= \frac{63}{\sqrt{63}\sqrt{63}} = 1$$

$$PC(w,z) = \frac{\sum_{i\in\{1,\ldots,6\}}(r_{ui} - 2.83)(r_{vi} - 2.83)}{\sqrt{\sum_{i\in\{1,\ldots,6\}}(r_{ui} - 2.83)^2}\sqrt{\sum_{i\in\{1,\ldots,6\}}(r_{ui} - 2.83)^2}} = 1$$

However, we have observed twice as much data in the second case – shouldn't our similarity score reflect more confidence in the similarity computation as we increase the amount of data we have available for input?

This question motivated us to derive a new similarity score, which we refer to as the Likelihood Ratio, or LiRa, similarity.

The LiRa score is described in detail in the following section, but we include the result of computing LiRa(u,v) here for comparison:

$$LiRa(x_u, x_v) \approx 1.19$$

$$LiRa(x_w, x_z) \approx 2.39$$

With twice as much data, the Lira score is twice as high.

Consider another example. Let:

$$x_a = \begin{bmatrix} 5 & 1 & - & - & - & 2 \end{bmatrix}, x_b = \begin{bmatrix} 1 & 1 & - & - & - & 2 \end{bmatrix} \tag{4.5}$$

Now, the CV, Pearson, and LiRa similarities for x_u and x_v are:

$$CV(a,b) = \frac{x_a^T x_b}{||x_a||||x_b||}$$

$$= \frac{10}{\sqrt{30}\sqrt{6}} \approx 0.7454$$

$$PC(a,b) = \frac{\sum_{i\in\{1,2,6\}}(r_{ai} - 2.67)(r_{bi} - 1.33)}{\sqrt{\sum_{i\in\{\}}(r_{ai} - 2.67)^2 \sum_{i\in\{1,2,6\}}(r_{bi} - 1.33)^2}}$$

$$= \frac{-0.67}{2.40} \approx -0.2774$$

$$\text{LiRa}(x_a, x_b) \approx 0.6887$$

Compare these scores to the similarities between two users c and d, for whom we observe more data. Notice that users a and b only have one difference of opinion in their ratings (for the first item), as is the case for users c and d.

$$x_c = \begin{bmatrix} 5 & 1 & 2 & 4 & 5 & 2 \end{bmatrix}, x_d = \begin{bmatrix} 1 & 1 & 2 & 4 & 5 & 2 \end{bmatrix} \tag{4.6}$$

Now, we have the following similarities:

$$CV(c,d) = \frac{55}{\sqrt{75}\sqrt{51}} \approx 0.8893$$

$$PC(c, d) \approx 0.5300$$

$$\text{LiRa}(c, d) \approx 1.8825$$

Note that the similarity is higher for users c and d than for users a and b in each case. However, with twice as much data, the LiRa score is more than twice as high for users c and d than for users a and b. In addition, unlike the Pearson similarity, the LiRa score is positive in both cases, indicating that there is evidence to suggest that both sets of users a and b, and c and d, have similar rating preferences, albeit there is much less evidence in the case of users a and b. In the next section, we elaborate on the intuition behind the LiRa score and derive the formula to compute the LiRa similarity between two users in a recommender system.

4.2 The LiRa Similarity Score

The idea to use a likelihood-based score for similarity computations in RS data was inspired in part by the *LOD score* popular in genetic mapping [5] and the concept of *modularity* in community detection [3]. In both cases, the concept of similarity is based on comparing the likelihood of the observed data, under some assumptions on the underlying data structure, to the likelihood of observing the data by chance. In genetic mapping, the LOD score relates the likelihood of observing genetic marker data, assuming genetic linkage, to the likelihood of observing the same genotypes by chance. Newman [3] introduced the idea that a community structure contains many more edges than expected if the edges among social network vertices were generated at random. Extending these ideas to the RS domain, we present the LiRa (**Li**kelihood **Ra**tio) Similarity.

4.2.1 Definition of the LiRa Similarity

For two discrete-valued vectors x_u and x_v, we define the Likelihood Ratio (LiRa) Similarity as follows:

$$\text{LiRa}(x_u, x_v) = \log_{10} \frac{p(\text{differences in } x_u \text{ and } x_v| \text{ same cluster })}{p(\text{differences in } x_u \text{ and } x_v| \text{ pure chance})} \tag{4.7}$$

where the numerator in the ratio is the probability of observing the values in x_u and x_v, assuming x_u and x_v belong to the same cluster in our cluster model, and the probability in the denominator is set by assuming that the entries in x_u and x_v were generated uniformly at random.

Suppose that the entries in each vector can take on only a finite number d of discrete values $\mathcal{V} = \{1, 2, \ldots, d\}$. Then, we can easily compute the probability that we observe the values x_{ui} and x_{vi} for co-observed entry i by chance, assuming that the values are generated uniformly and independently at random. This is probability is simply $\frac{1}{d^2}$. Therefore, the probability that the two vectors match exactly in a particular entry i is $p(|x_{ui} - x_{vi}| = 0) = \frac{d}{d^2} = \frac{1}{d}$. Similarly, we can easily derive $p(|x_{ui} - x_{vi}| = \delta)$ for $\delta = 1, ..., d-1$. The denominator in the ratio in LiRa is thus defined as:

$$p(\text{differences in } x_u \text{ and } x_v| \text{ pure chance}) = \prod_{\delta=0}^{d-1} b_\delta^{\#\delta} \tag{4.8}$$

where $b_\delta = p(|x_{ui} - x_{vi}| = \delta)$, assuming that x_{ui} and x_{vi} were generated by a uniform distribution over the values $\mathcal{V}$, and $\#\delta$ is the number of times that we observe a difference of δ in the co-observed entries.

The challenge is to define the probability of observing a difference of δ in the values x_{ui} and x_{vi}, under the assumption that x_u and x_v belong to the same cluster. This is not a trivial task, and we argue that this model will be dependent on the application of interest.

For RS data, we make two assumptions that we believe lead to one plausible model: (1) An underlying cluster structure exists in RS data: There exist a set of clusters $\mathcal{C}_1, ..., \mathcal{C}_\kappa$ such that each user u belongs to at least one $\mathcal{C}_c$, and (2) The probability distribution on differences in user ratings is fixed within a cluster, with a greater probability of observing matching than mismatched ratings. These assumptions encapsulate the intuition that similar users tend to rate items similarly.

With these assumptions, we define the following probability distribution over the differences $|x_{ui} - x_{vi}|$:

$$c_\delta = p(|x_{ui} - x_{vi}| = \delta | \text{ same cluster}) = \left(\frac{1}{2}\right)^{\delta+1} \tag{4.9}$$

with the exception that

$$c_{d-1} = p(|x_{ui} - x_{vi}| = d-1) = 1 - \sum_{\delta=0}^{d-2} c_\delta = \frac{1}{2^{d-1}} \tag{4.10}$$

to ensure a proper probability distribution. Therefore the numerator in the ratio in LiRa becomes:

$$p(\text{differences in } x_u \text{ and } x_v | \text{ same cluster }) = \prod_{\delta=0}^{d-1} c_\delta^{\#\delta} \tag{4.11}$$

where c_δ and $\#\delta$ are defined above, and $x_{ui} = r_{ui}$ if user u rated item i, and $-$ otherwise, where $-$ indicates a missing value.

We emphasize that both x_u and x_v may have many missing values, which are not taken into account when evaluating these probabilities. In particular, the values are not simply treated as 0's as in the Cosine Vector similarity score. On the other hand, as long as $\frac{1}{2} > \frac{1}{d}$, the LiRa score increases with a greater number of matching co-observed entries, and in general the contribution to the LiRa score of the rating difference for a co-observed item i will depend on the number of discrete values d. For example, with

$d = 5$, $b_1 > c_1$, but at $d = 10$, $b_1 < c_1$, thus a difference of 1 in a rating scale of 1 to 5 will decrease the LiRa score, whereas on a rating scale of 1 to 10, a difference of 1 in user ratings will increase it. To see this, notice we can re-write the LiRa score as:

$$\sum_{d=1}^{\delta} (\#\delta) \log_{10} \left(\frac{c_\delta}{b_\delta} \right) \tag{4.12}$$

and thus $\log_{10}(c_\delta / b_\delta)$ is the amount that a pair of co-observed ratings x_{ui} and x_{vi} with a rating difference of δ will contribute to the similarity score.

In future work, we plan to explore other, perhaps more plausible, multinomial probability distributions over the differences in user ratings that capture the intuition that users in the same cluster should rate items with very close rating values. One possible extension to LiRa, that takes a more data-driven approach, is to fit the data to a model that more accurately captures how ratings are typically distributed, instead of assuming uniformly generated data.

However, we claim that this simple model captures enough of the intuition that users with similar preferences are more likely to agree than disagree in their ratings of the same item. We will show in Section 4.3 that these assumptions lead to a useful similarity score for RS data.

We conclude with an example using the same the vectors y_u and y_v from Equation 4.3. The corresponding vectors x_u and x_v are:

$$x_u = \begin{bmatrix} 1 & 1 & - & - & - & 2 \end{bmatrix}, x_v = \begin{bmatrix} 1 & 1 & - & - & - & 2 \end{bmatrix}$$

Suppose that there are $d = 5$ discrete rating values in the data set. We get the LiRa similarity:

$$\text{LiRa}(x_u, x_v) = \log_{10} \frac{\left(\frac{1}{2}\right)^3}{\left(\frac{1}{5}\right)^3} = 1.19 \tag{4.13}$$

Now consider $\text{LiRa}(x_u, x_v)$ when we observe the full vectors:

$$x_u = \begin{bmatrix} 1 & 1 & 5 & 4 & 4 & 2 \end{bmatrix}, x_v = \begin{bmatrix} 1 & 1 & 5 & 4 & 4 & 2 \end{bmatrix}$$

Now, we have:

$$\text{LiRa}(x_u, x_v) = \log_{10} \frac{(\frac{1}{2})^6}{(\frac{1}{5})^6} = 2.39 \tag{4.14}$$

With twice as much data, the LiRa similarity is twice as high. Note that, in particular, the maximum LiRa score for any two vectors is always attained when the two vectors are equal, but that the similarity grows as $O(n \log_{10} d)$, where n is the dimensionality of the input vectors and d is again the number of discrete rating values. Contrast this with the Pearson or Cosine similarities, which will attain a maximum of 1, regardless of the amount of data observed.

Like modularity in community detection and the LOD score in genetic mapping, the LiRA similarity makes assumptions on the underlying clustering structure in the data in order to better evaluate similarity among entities in the data.

4.3 Empirical Evaluation

We evaluate the effectiveness of the LiRa similarity in comparison to other similarity scores in RS data in two ways: (1) We compare the prediction accuracy of a simple kNN method using various similarity scores on real data sets, and (2) We evaluate the ability of several similarity scores to distinguish points within the same cluster from points in different clusters in synthetic data. Experiments on real data sets show that the LiRa similarity can detect similar users in a realistic setting with better accuracy than other scores. The synthetic data allows us to observe the effect of missing entries and dimensionality on similarity computations, and to verify that the LiRa score detects

users from the same cluster when a known clustering exists within the data.

4.3.1 Data

As Herlocker et al. note in their overview of methods for evaluating recommender systems [79], there are few publicly available data sets that can be used to test hypotheses about RS data, forcing most research in this field to experiment on the few available data sources. Our source of real data are the publicly available MovieLens data sets, which are among the most often referenced data sets in RS literature [80]. Here, we report results on the 100K and 1M MovieLens data sets[1]. For the 100K dataset, we used the u1-u5 *.base* and *.test* sets when evaluating prediction accuracy. For the 1M dataset, we randomly split the original rating data into five sets of 80%/20% training/test pairs.

In addition to our empirical evaluation on real data, we generated a small set of synthetic data sets. The publicly available data sets we are aware of are not rich enough to examine the effects we are intersted in evaluating – most already contain very high missing rates, thus simply deleting existing entries to simulate more missing data would result in a very limited range of test cases for experimentation. Our experiments on synthetic data give us an in-depth view of the effects of missing entries in RS data.

4.3.2 Experiments on Real Data

We first give the pertinent details of our implementation of the kNN algorithm, which is used to evaluate the effectivenes of a similarity score in detecting users with similar preferences. For each rating r_{ui} that user u gave to item i in the test set, we first find at most k nearest neighbors of user u in the training set, among those who rated item i. Each neighbor v of u has the property that the similarity $S(u, v)$ is greater than or

[1] `http://grouplens.org/datasets/movielens/`

equal to $S(u, z)$ for any other user z in the training set, where $S(u, v)$ is the similarity score between u and v in the training set. The number of neighbors is less than k if less than k users rated item i in the training set. Next, the prediction p_{ui} of the rating that user u gives item i is computed by taking an unweighted average of the ratings that the neighbors of u have item i.

The Root Mean Squared Error (RMSE) "is perhaps the most popular metric used in evaluating accuracy of predicted ratings" [81]. Another popular measure of prediction accuracy is the Mean Absolute Error (MAE). The RMSE and MAE are defined as follows:

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{r_{ui} \in \mathcal{T}} (r_{ui} - p_{ui})^2} \ , \ MAE = \frac{1}{|\mathcal{T}|} \sum_{r_{ui} \in \mathcal{T}} |r_{ui} - p_{ui}|$$

where $\mathcal{T}$ is the test set of ground truth ratings. We report both the RMSE and MAE for ratings predicted by the kNN method for various values of k and several similarity scores.

In addition to the Pearson, Cosine and LiRa scores, we evaluate the kNN prediction accuracy using Patra et al.'s recently proposed BCF score [12]. The Bhattacharyya coefficient for collaborative filtering (BCF) attempts to use global item similarities as weights in local user rating similarity computations, and was reported to perform well on extremely sparse data sets. It is defined as follows:

$$\text{BCF}(x_i, x_j) = \text{Jacc}(x_i, x_j) + \textstyle\sum_{i \in I_u} \sum_{j \in I_v} BC(i, j) \text{loc}(x_{ui}, x_{vj}) \tag{4.15}$$

where

$$BC(i, j) = \sum_{\rho=1}^{d} \sqrt{\frac{\#\rho_i}{\#i} \frac{\#\rho_j}{\#j}}$$

where d is the number of rating values, $\#i$ is the number of users that rated item i, and $\#\rho_i$ is the number of users that rated item i with value ρ. I_u, I_v, and I_{uv} are defined as

in Section 4.1. Thus BC gives more weight to the local similarity $\text{loc}(x_{ui}, x_{vj})$ if items i and j have similar rating distributions across users in the entire training set. $\text{loc}(x_{ui}, x_{vj})$ is a local similarity measure of the ratings that user u gave to item i and user v gave to item j. Of the two *loc* similarity scores defined by Patra et al., we chose to use loc_{corr}, defined as:

$$\text{loc}_{\text{corr}}(x_u, x_v) = \frac{(x_{ui} - \bar{x}_u)(x_{vj} - \bar{x}_v)}{\sigma_u \sigma_v}$$

where σ_u is the standard deviation of ratings made by user u and $\bar{x}_u$ is the mean of ratings made by user u. In the experimental evaluation of Patra et al., loc_{corr} achieved lower rating prediction error than their other *loc* similarity score. $\text{Jacc}(x_u, x_v)$ is the Jaccard similarity:

$$\text{Jacc}(x_u, x_v) = \frac{|I_{uv}|}{|I_u| + |I_v|}$$

4.3.3 kNN Results

The MAE and RMSE for kNN prediction on the MovieLens 100K data sets are shown in Figure 4.2, with the MovieLens 1M MAE results in Figure 4.1. Note that the LiRa similarity outperforms other similarity scores in prediction accuracy and for a wide range of choices for the number of neighbors k. We include both the MAE and RMSE results for the 100K data sets, but omit the RMSE for the 1M data sets due to space constraints. However, the RMSE results for the 1M data sets show the same trend as is seen in the MAE results – that is, LiRa dominates the other similarity scores in accuracy, and the gap between LiRa and other similarity scores' prediction error widens both when we increase the size of the data set, and when we decrease k.

The kNN curves have the expected shape – at low values of k, the data is being under-utilized, because there are on average more than k users who rated item i and are truly similar to u in the data, but they are being left out of the computation of the

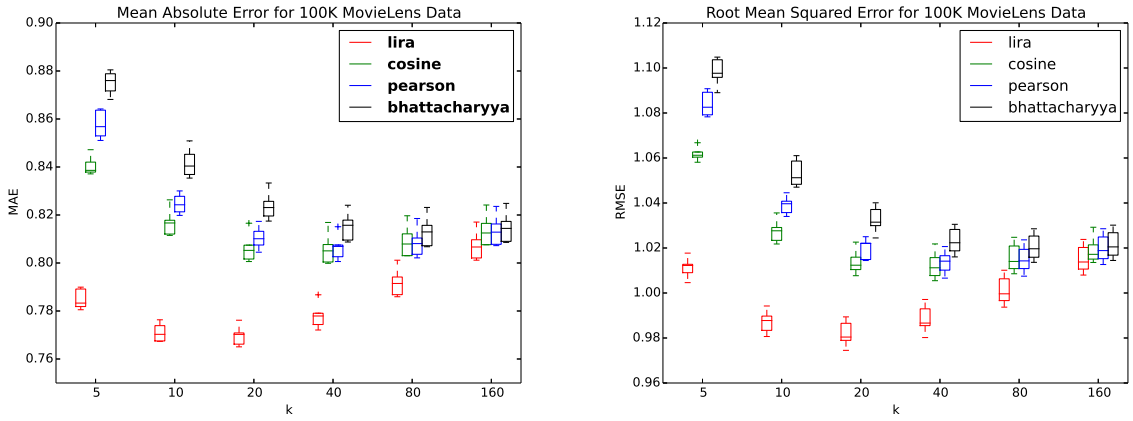

Figure 4.2: Comparison of prediction accuracy using various similarity scores in kNN show LiRa's better ability to choose relevant neighbors.

prediction p_{ui}. At the other extreme, at very high values of k, there are on average less than k truly similar users to u in the data who rated item i, and the additional users in the neighbor set are not useful in predicting u's rating of item i.

However, the best value of k for LiRa tends to be lower than the best value of k using other similarity scores, and the LiRa score outperforms other similarity scores for all values of k where the neighbor set is not so large that it is virtually the same for each similarity score (at $k = 160$, the number of neighbors is 17% of the 100K training set size, meaning that for most users, the prediction p_{ui} is based on *all* users in the training set who rated item i). In addition, as k decreases, the gap between LiRa and the other scores' error widens. From these observations, we conclude that LiRa is better able to distinguish between truly similar and truly dissimilar users; for a given k, it finds a better set of k neighbors than the other scores, and as k decreases, it keeps more of the neighbors that are the better predictors of the rating in the neighbor set than the other scores.

The Shrinkage Factor

One approach that has previously been found to improve rating prediction accuracy of the Pearson correlation coefficient in practice is the use of a *shrinkage factor*. A shrinkage factor β is used to decrease the weight of a similarity score that is based on few co-rated items in the user-based collaborative filtering setting, or few co-rating users in the item-based collaborative filtering setting. The Pearson similarity s of users u and v is adjusted to a modified similarity $\hat{s}$ with the β parameter as follows:

$$\hat{s} = \left(\frac{\eta_{uv} - 1}{\eta_{uv} - 1 + \beta} \right) s$$

where η_{uv} is the number of co-rated items between users u and v.

Thus if users u and v have co-rated much fewer items than the constant β, the similarity between these two users will be shrunk toward zero. If, on the other hand, users u and v have co-rated a number of items much larger than β, then shrinkage of the original similarity score s will not be significant. The shrinkage factor can be understood from a Bayesian perspective, as described by Koren and Bell in their study *Advances in Collaborative Filtering* and Chapter 3 of the Recommender Systems Handbook [82].

We tested the shrinkage factor in combination with the Pearson similarity score in order to evaluate its usefulness, as compared to LiRa, in finding truly similar neighbors in the $k$nn setting for the MovieLens100K data set. We used a shrinkage factor of $\beta = 25$, since this setting of β has been shown to produce the optimal results in a user-based collaborative filtering setting, using the Pearson similarity score[2]. In addition, we also experimented with the use of the shrinkage factor in combination with the LiRa score, although we do not expect to see significant improvement over the raw LiRa score in this case, since LiRa was designed to address the discrepancy between many and few co-rated

[2]See results here: `http://www.mymedialite.net/examples/datasets.html`

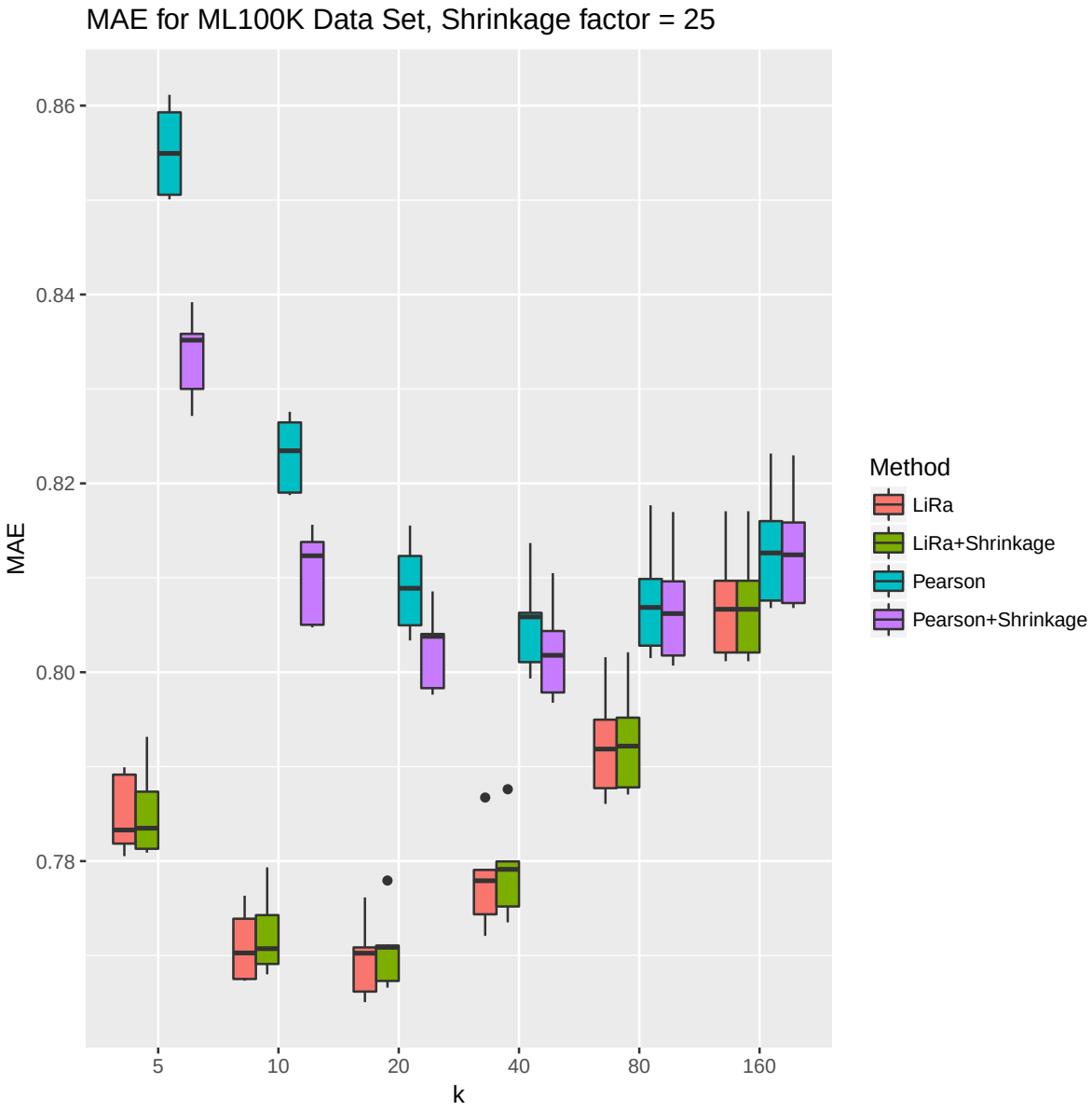

Figure 4.3: The shrinkage factor method improves the prediction accuracy of a user-based $k$nn rating prediction method using the Pearson similarity score. However, this improvement is insufficient to compete with the prediction accuracy when the LiRa similarity score is used instead. Using shrinkage in conjunction with the LiRa score does not appear to improve prediction accuracy over a pure LiRa-based approach.

items.

Figure 4.3.3 shows how using the shrinkage factor in conjunction with the Pearson correlation similarity improves the MAE of predicted ratings using a user-based k-nearest neighbor approach. However, the plot also demonstrates that despite the usefulness of the shrinkage factor in reducing the effect of few co-rated items between users, using the LiRa score is still significantly more effective at finding useful neighbors. As expected, using the shrinkage factor in combination with the LiRa score does not significantly improve results.

We postulate that these results demonstrate LiRa's ability to take into account the amount of data that is being used to evaluate a similarity score for two users in the data set in order to make a better determination of similarity. To test this hypothesis, we next present experiments on synthetic data, where we can control the missing data rate and the "true" similarity among users.

4.3.4 Experiments on Synthetic Data

Our goal in generating synthetic data was to evaluate LiRa's behavior for increasing missing data rates in a setting that resembles RS data, but where we can control and understand the underlying similarities of users in the data set. Therefore we use a very simple generative model to produce two clusters $\mathcal{C}_1, \mathcal{C}_2$ of $m/2$ users each, where each cluster contains users with similar rating patterns on n items and d discrete rating values. For all experiments in Section 4.3.5, we set m to 40, d to 5, and varied n to examine the effects of increasing dimensionality and user-to-item ratio. We used the following procedure to generate the two clusters:

1. For each of the two clusters $\mathcal{C}_1, \mathcal{C}_2$, and for each of the n items i, randomly choose a d-dimensional parameter $\mu_{ki} \in \mathcal{R}^d$, which defines the multinomial distribution

$f_{ki}(x = \rho|\mu_{ki})$ over the d rating values as:

$$f_{ki}(x = \rho|\mu_{ki}) = \mu_{ki}(\rho)$$

2. For each user u and item i in each cluster $\mathcal{C}_\kappa$, generate rating x_{ui} from $f_{ki}(x_{ui} = \rho|\mu_{ki})$

Thus users from the same cluster will tend to have similar rating patterns. To illustrate our simple model we provide the following example: suppose we set $m = 40$, $n = 2$, and $d = 5$. Our simulation produces the following 5-dimensional parameters $\mu_{ki} \in \mathcal{R}^5$:

$$\mu_{11} = [0.55, 0.09, 0.25, 0.01, 0.10]\,, \mu_{12} = [0.34, 0.33, 0.29, 0.03, 0.01]$$

$$\mu_{21} = [0.17, 0.08, 0.12, 0.33, 0.30]\,, \mu_{22} = [0.04, 0.25, 0.47, 0.07, 0.18]$$

Therefore users in cluster $\mathcal{C}_1$ tend to rate item 1 about half the time with a value of 1 (since $\mu_{11}(1) = 0.55$) about a quarter of the time with a value of 3 ($\mu_{11}(3) = 0.25$), and much less often with a value of 2, 4, or 5. The same users in cluster 1 tend to rate item 2 with a value of either 1,2, or 3, and much less often with a value of 4 or 5. Therefore, one likely user u from C_1 can be represented by the vector $x_u = [1, 2]$, whereas a likely user from C_2 is $x_v = [4, 2]$.

Note that the parameters μ_{ki} are generated at random, but sum to one and are consistent within a cluster, ensuring that users from the same cluster rate items with the same patterns. Therefore, we expect that the similarity between two users within the same cluster should be high when compared to the similarity of two users in different clusters. However, we did not explicitly generate the idealized clusters that make up the model used in our LiRa score, to show that the oversimplified model used by LiRa is nevertheless enough to capture much of the intra-cluster similarity and inter-cluster

dissimilarity.

We quantify a similarity score's ability to resolve two users in the same cluster from two users in different clusters with a quantity we call the score's *resolution*. The resolution of a score S is defined as the mean of $S(x_u, x_v)$ for all points x_u and x_v within the same cluster minus the mean of $S(x_u, x_w)$ for all points x_u and x_w in different clusters. Note that the resolution defined here is not the fundamental resolution defined in Chapter 3. In experiments we set S to the LiRa, Pearson, Cosine, and Bhattacharyya similarity scores, for increasing values of the dimensionality n. A positive resolution value indicates greater average intra- than inter- cluster similarity, meaning that a score S is greater for points in the same cluster than points in different clusters on average. To additionally observe the effect of missing data on the similarity scores, we randomly deleted an increasing fraction of the ratings x_{ui}. Thus the expected number of co-observed entries decreases with the missing rate.

4.3.5 Results on Synthetic Data

The scaled resolution is plotted in Figure 4.4 for the LiRa (red), Pearson (blue), Cosine (green) and Bhattacharrya (black) similarities. Each marker in each plot corresponds to a different dimensionality n, where n increases from 5 to 80, doubling each time, and the missing rate increases from 0.1 to 0.9 in increments of 0.1, with an additional point at 0.95. We scaled each score's resolution by dividing by the maximum-magnitude resolution achieved by that score in the experiments.Therefore, a scaled resolution of 1 indicates the missing rate and dimensionality with the highest-magnitude resolution over all missing rates and dimensionalities, and a magnitude less than one tells us what fraction of the maximum-magnitude resolution was achieved at a particular dimensionality and missing rate.

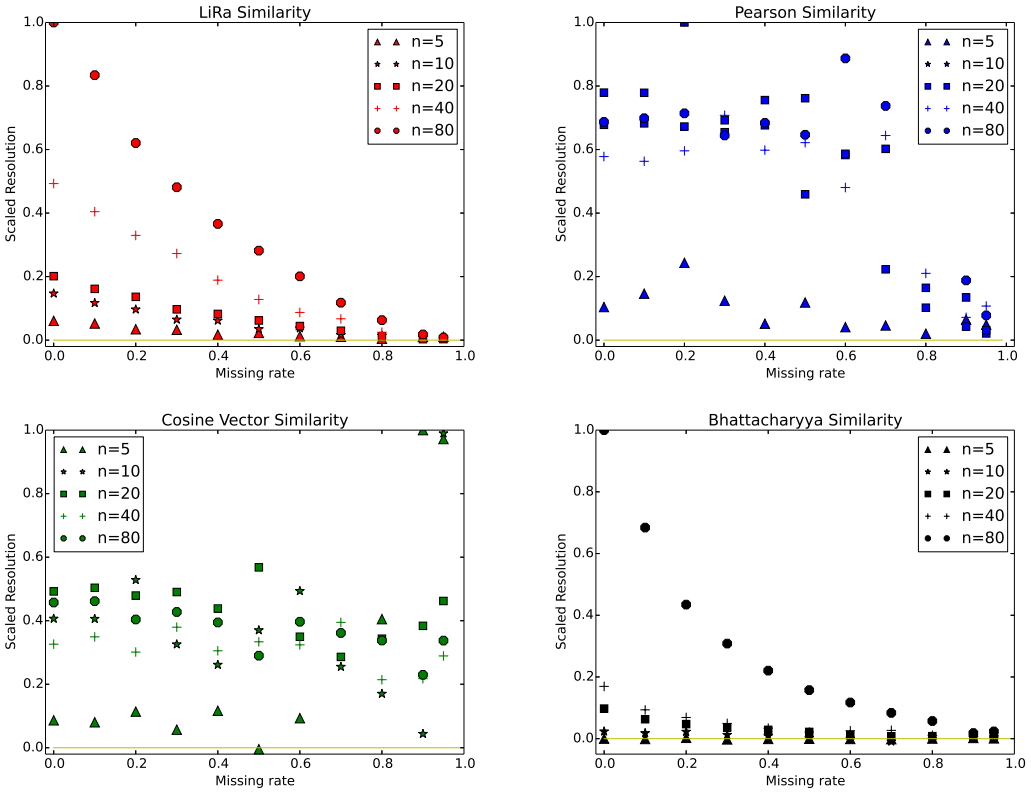

Figure 4.4: Similarity resolution (higher is better) as a function of missing data rate plotted for four similarity scores: LiRa (upper left), Pearson (upper right), Cosine (lower left), and Bhattacharyya (lower right). Resolution indicates a score's ability to differentiate a pair of points in the same cluster from a pair of points in different clusters. Cosine and Pearson scores do not improve in resolution with more data availability.

We observe that the resolution of the LiRa score is greater as we decrease missing data and increase the dimensionality. In addition, the LiRa resolution is positive for all values of the missing rate and all dimensionalities (the minimum of LiRa resolutions was 0.051), indicating greater average intra- than inter-cluster LiRa similarity. The greater magnitude of the resolution in the presence of more data shows the ability of LiRa to make stronger claims about similarity as the number of co-observed entries in two discrete-valued vectors increases. More data comes in the form of a lower missing data rate, but also increased dimensionality, because there will be more expected co-observed

entries between two vectors when the dimensionality is higher.

Contrast this with the Pearson and Cosine similarity scores, where dimensionality and missing data have virtually no effect on the resolution, except that high values of missing rates tend to lower the Pearson resolution dramatically. The resolution is positive for most values of the dimensionality and missing rates for Cosine, and all values of dimensionality and missing rates for Pearson, but increasing the amount available data does not improve the ability of Cosine or Pearson to resolve similar from dissimilar users. The fact that low dimensionalities and higher missing rates often yield a higher resolution than higher dimensionalities and lower missing rates shows theses scores' inability to make use of more data for more accurate similarity judgments. For the most part, the Bhattacharyya resolution tends to increase with increasing dimensionality and decreasing missing rates, also indicating a greater difference between intra- and inter- cluster scores, but there are instances when this is not the case.

We conclude this section with a discussion of Figure 4.5, which plots the scaled average inter-cluster similarity score across missing rates for the four tested scores, with a fixed dimension of $n = 80$. Scaling was again done by dividing each average inter-cluster similarity by the magnitude of the greatest-magnitude average inter-cluster similarity that occurred over all missing rates. This way, we can see how inter-cluster similarity changes with the missing rate, for all scores on the same scale.

Recall that a negative Pearson or LiRa value indicates dissimilarity in some way. A negative Pearson score indicates anti-correlated co-observed entries. A negative LiRa score indicates a greater chance that the data in co-observed entries was generated by chance, rather than that the data comes from two vectors in the same cluster. In Bhattacharyya, a negative score also indicates anti-correlation in the co-observed entries, but weighted by item similarity and shifted by the Jaccard similarity, making it harder to interpret. Cosine is restricted to positive values in this setting, because all vector entries

were positive.

We observe that LiRa and Pearson appear to be better able to indicate dissimilarity, as their scaled average inter-cluster values are negative for most and all missing rates, respectively. However, LiRa again makes use of more data to make a stronger claim about dissimilarity, giving greater-magnitude negative values when more data is observed. Both Cosine and Bhattacharyya remain positive, indicating some similarity between points from different clusters, and actually increase in magnitude as more data becomes available, scoring two points from different clusters higher at lower missing rates. Based on the MovieLens results, this may also mean that the real data contains user clusters, and the Bhattacharyya score is too high for users from different clusters. Thus it may choose sub-optimal neighbors in the kNN method in this case.

Based on our results from these synthetic data experiments, we make two concluding remarks about the results on real data in Section 4.3.3. First, the high performance of LiRa on the MovieLens data can be explained by its dependence on not only the rating patterns in co-observed entries of user rating vectors, but also on the amount of data that is available to make the similarity computation. Second, based on the Bhattacharyya results, we believe a promising future research direction is to further investigate what type of underlying cluster structure exists in RS data. The intuition behind the Bhattacharyya similarity is that a higher similarity between two items in the set I_{uv}, defined by the difference between their rating distributions across the entire data set, should contribute a higher weight to the difference in these item ratings. Each of the $|I_{uv}|^2$ pairs of co-rated items (i, j) contributes to the similarity score. By contrast, LiRa does not consider item similarity in the computation of user similarity, and only examines the differences in user ratings of the $|I_{uv}|$ co-rated items. A clustering structure such as that assumed by LiRa may indeed exist in real-world data, and perhaps the Bhattacharyya score is not well suited to this setting, where it can be high for users from different clusters, despite its

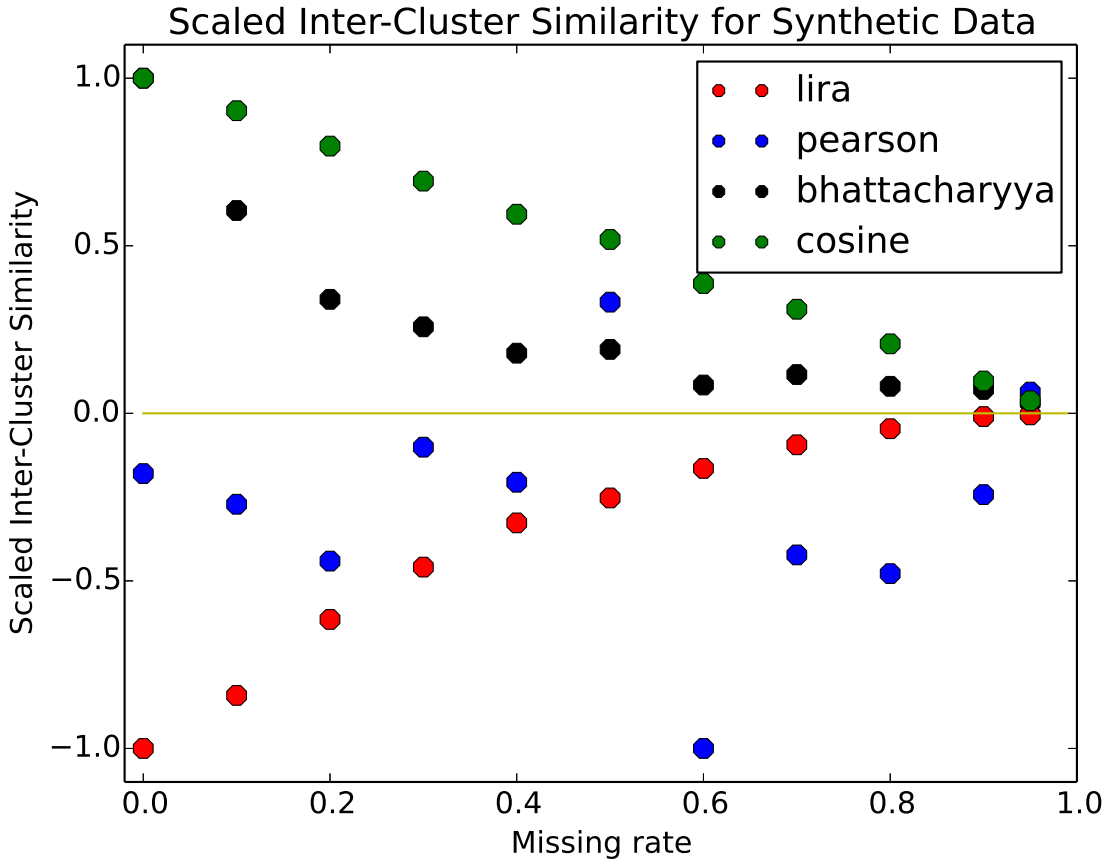

Figure 4.5: The inter-cluster similarity of various similarity scores on synthetic data. LiRa is the only score which decreases with decreasing missing rate, indicating greater ability to differentiate points from different clusters when more data is available.

ability to give greater scores to users from the same cluster with more data.

4.4 Related Work

4.4.1 The LLR Similarity

We devote a subsection to discuss the LLR similarity, which has been useful in the industrial setting when evaluating similarity between unary data in the recommender system setting.

In the context of statistical text analysis, Dunning [83] makes a strong case for the use of a log-likelihood ratio to examine the statistical significance of word or bigram frequencies in sparse data. Dunning's work has been applied to the recommender systems domain through the use of a score known as the LLR similarity. The LLR can be applied as a similarity score to unary data, where the only available data is whether each user rated, bought, or otherwise interacted with an item – each user either interacted with an item or did not. However, in the more general setting, the LLR acts as a filtering mechanism rather than a similarity score for users or items. The LLR is used as a test of statistical independence between rated and unrated items in either a user-based or item-based approach.

For example, suppose we have the following rating vectors in $n = 22$ dimensions for users u and v:

$$x_u = \begin{bmatrix} 3 & - & - & 4 & - & - & - & 1 & - & - & 5 & - & - & - & 4 & - & - & 4 & 4 & - & - & 2 \end{bmatrix}$$

$$x_v = \begin{bmatrix} 5 & - & - & - & - & - & - & 5 & 1 & - & 5 & - & - & - & 1 & - & - & - & 5 & - & - & 2 \end{bmatrix}$$

LLR compares u and v by first evaluating the contigency table of rated and unrated

items by u and v, as follows:

	rated by v	not rated by v
rated by u	6	2
not rated by u	1	13

If we let k represent the contingency table, where k_{ij} is the ith row and jth column of the table, we have:

$$LLR(x_u, x_v) = 2(\sum_{i,j} k_{ij})(H(k) - H(\text{rowSums(k)}) - H(\text{colSums(k)}))$$

where H is Shannon's entropy, computed as the sum of (k_{ij} / sum(k)) log (k_{ij} / sum(k)).

Thus for the example we have:

$$LLR(x_u, x_v) = 2(22)(-0.2650) \approx 11.3193$$

The LLR similarity in this case is high, and indicates that it is unlikely that we would see the number of rated and unrated items in Table 4.4.1 if users u and v independently chose the items for which to give ratings. In other words, there is evidence to reject the hypothesis that users u and v independently chose the items that they will rate.

Compare this to LiRa, which is computed using only the (six) entries that were rated by both u and v, and in this case evaluates to:

$$\text{LiRa}(x_u, x_v) = \log_{10}\left(\frac{(\frac{1}{8})(\frac{1}{16})(\frac{1}{2})(\frac{1}{16})(\frac{1}{4})(\frac{1}{2})}{(\frac{6}{25})(\frac{2}{25})(\frac{1}{5})(\frac{4}{25})(\frac{8}{25})(\frac{1}{5})}\right) \approx -0.1101$$

Unlike the LLR similarity, the LiRa similarity for x_u and x_v is negative, suggesting that the users do not exhibit similar rating patterns. A negative LiRa score indicates that it is less likely that users u and v belong to the same cluster, using the simple clustering model described in Section 4.2, than that the rating differences we observe are due to

pure chance. While the LLR indicates that users u and v choose highly overlapping sets of items to rate, it does not tell us whether u and v have similar tastes.

Again, it is important to note that LLR is not a similarity score but rather acts as a filter when finding nearest neighbors. Indeed, the author himself states: "I recommend using large LLR as a mask and then estimating the strength of similarity by other means."[3]

The LiRa ratio expresses how likely it is that two users are similar, based on the assumption that user similarity is reflected by the probability of two users' ratings being distributed according to the clustering model presented in Section 4.2.

By contrast, the LLR similarity is a likelihood ratio test that is used to test whether or not the rating patterns of two users are independent. If the LLR is high, it indicates that there is sufficient evidence to reject the hypothesis that two users rate independently. That is, a high LLR similarity between users u and v suggests that there is a correlation between the items that u chooses to rate and the items that v chooses to rate. Note, however, that the LLR similarity does not take into account the values of the ratings. Thus, as in the example above, a high LLR does not necessarily indicate a high similarity between two users' rating preferences.

Dunning's work has been adapted to RS data in several ways and has been shown to enhance the performance of recommender systems in an industrial setting [84, 85]. However, we emphasize that in the applied versions of Dunning's likelihood ratio to RS data, the ratio is used either as a filter for finding relevant items to use in similarity computations, or a weighting term in the rating prediction phase, and has not been developed into a similarity score that depends on *rating values*.

[3] `http://tdunning.blogspot.com/2008/03/surprise-and-coincidence.html`

4.4.2 Other Approaches to Evaluating Similarity in Recommender Systems

Early studies [13, 41, 40] consistently rate the Pearson, Cosine Vector, or slight variants as the superior similarity scores for RS data. Several recent studies [42, 12] focus on the cold start problem, in which extremely few ratings are available for a new user, making it difficult to determine her similarity to other users with traditional similarity scores.

The PIP heurisitic was introduced by Ahn [42] to address the cold-start problem, but was shown to perform comparably to traditional similarity scores such as Pearson's correlation coefficient in non-cold start settings, and was outperformed in terms of rating prediction accuracy by the Bhattacharyya coefficient for collaborative filtering in cold start settings with extremely few ratings. Patra et al.'s Bhattacharyya coefficient for collaborative filtering, defined in Section 4.3, takes into account item similarity as a weighting scheme for user similarity. It was developed for the extremely sparse setting, thus Patra et al.'s empirical evaluation of its use in rating prediction accuracy is restricted to data sets where the missing rate is much higher than even the already sparse MovieLens datasets, and is better suited for the cold-start setting. The Bhattacharyya coefficient also is more computationally expensive than our LiRa score, as it requires all-to-all user-to-user as well as all-to-all item-to-item similarity computations.

Our method also slightly resembles Jojic et al.'s [86] item similarity score, where similarity is determined by comparing the number of users that like two items to the number expected by chance. However, their method is limited to binary like/dislike data, treats dislikes the same as missing entries in the item similarity computation, and was used in combination with additional heuristics to determine whether a user will like a particular item.

The intuition that inherent clusters of users exist in RS data has been explored by several clustering methods which were developed to improve prediction accuracy in RS data. Sarwar et al. [87] used clustering to improve scalability by first partitioning the users into clusters, then making a prediction based on averaging ratings from members of the same cluster, allowing for less computation time than a kNN method on the MoveLens 100K data set. Similarly, Xue et al. introduce a k-means clustering phase prior to prediction [88], and predict a rating for a user u by choosing k neighbors out of the clusters with *representatives* that score highly with u. Rashid et al. [89] incorporate bisecting k-means clustering "to increase efficiency and scalability while maintaining good recommendation quality" in their ClustKNN algorithm. Das et al. [90, 91] use a DBSCAN-based algorithm to improve kNN prediction accuracy. An extensive experimental study of the effectiveness of various centroid selection methods for the k-means algorithm when used as a pre-processing step in recommendation systems is presented by Zahra et al. [92], who conclude that although many approaches improve prediction accuracy and efficiency, no algorithm is "a panacea" across all data sets. Nonetheless, the results of clustering-based approaches in rating prediction are encouraging, as they show promise in improving both accuracy and scalability of recommender systems.

4.5 Conclusion

We have introduced the LiRa similarity score for discrete-valued, sparse, and high-dimensional data, typical of the RS domain. We have shown through empirical evaluations on both real and synthetic data that LiRa's assumptions about a clustering model of users makes it a good indicator of user similarity, and that it outperforms other measures in this capacity. Although we have not evaluated against item-based collaborative filtering here, which has in some cases been shown to be superior to its user-based coun-

terpart in terms of rating prediction accuracy and efficiency [13], we believe that our focus on user-based collaborative filtering is justified. User-based collaborative filtering may produce more *novel* recommendations [93], for example, which is a desirable quality in state-of-the art recommender systems [94]. In addition, efficiency of user-based approaches can be further enhanced by techniques such as cluster-based smoothing [88].

An exciting area to focus our future research is exploring how to devise a better model of clustering structure within RS data in order to improve prediction accuracy of collaborative filtering methods. A first step in this direction would be to evaluate the *clusterability* [95] of RS data, which would provide insight as to the strength of clustering tendency in the data set. Another possible research direction is to develop fast clustering methods that use LiRa as a similarity score to improve the scalability of user-based collaborative filtering.

Chapter 5

Conclusion

I have focused on simultaneously addressing the large scale and missing value challenges to knowledge discovery in discrete-valued data, by leveraging inherent structural properties of underlying clusters in the data set. In the preceding chapters, I have focused on two application domains where large-scale data is commonplace and missing data is abundant.

I have introduced the concept of a fundamental resolution in large-scale, binary-valued sparse data, and designed fast clustering algorithms for genetic data sets with a low fundamental resolution. I have also developed a new similarity score for recommender system data that assumes an underlying clustering structure in the data set.

The BubbleCluster algorithm is described in Chapter 2. BubbleCluster clusters high-dimensional, sparse, and binary-valued input data of m points in $O(m\log(m)+mr)$ time, where r is the number of sketch points (see Section 25), and has been shown outperform the $O(m^2)$ running time of state-of-the-art genetic mapping tools. The clustering algorithm takes one pass over sorted data to build a sketch of the structure of each cluster, with linked sketch points forming a chain of bubbles that represent areas of high linkage probability. Once the sketches are formed, remaining points are assigned to clusters based on a high log-odds score, known in genetics as the LOD score, with established sketch points. We accurately clustered real-world, large-scale genetic map data, and showed that theoretically and experimentally that our approach was scalable. Our work has also

enabled the map construction of the grand-challenge, 1.6 million-marker wheat genome [64]. This work demonstrates the importance of carefully considering the underlying clustering structure of a data set to the design of efficient clustering algorithms.

As a next step toward understanding and efficiently analyzing large-scale genetic map data, the data reduction method described in Chapter 3 exploits a property of such data that we name its *fundamental resolution* in order to quickly find a set of representative points that characterize the entire data set. Because homozygous genetic map data is binary-valued and because a natural order exists in points within the same cluster, the large set of input points often contains a lot of redundant information. We can therefore model the data as many samples from a small-scale, more complete and accurate set of points that represent the genetic map. Once the large-scale input data is reduced to this set of points, the genetic map can be much more easily inferred, as demonstrated in our up-stream analysis of genetic map construction using only the representative points as input (see Section 3.4.7). To the best of our knowledge, no previous work has defined or exploited the fundamental resolution of genetic map data in this way. As was the case in the clustering work described in Chapter 2, insights into the particular structure of clusters in genetic map data led to the design of a fast, accurate method for data analysis in Chapter 3. We note that our data reduction algorithm achieved linear speedup empirically, even with high missing value rates and errors in the input data, potentially empowering genetic mapping tools to process unprecedented amounts of high-throughput sequence data.

The last portion of my dissertation has focused on exploring a generalization of the concept of fundamental resolution to a broader class of discrete-valued data with missing values, with an application focus on recommender systems. My work on a log-odds based similarity score for recommender systems data, that we call the LiRa score, is described in Chapter 4. LiRa is based on the assumption that in recommender systems, the set of

users can be clustered to a fundamental resolution of groups which represent very similar rating patterns. We have shown that LiRa is effective in finding neighbors of a user that are more useful than those found using the standard Pearson or Cosine similarity scores when using a k-nearest neighbor approach to user-based collaborative filtering. The simple k-nearest neighbor classification method is still a widely used technique in collaborative filtering [96], and has certainly been extended to enhance rating prediction accuracy with approaches such as filtering [97], clustering [88], or careful weighting schemes [98]. However, we note that in terms of rating prediction accuracy, the raw LiRa score outperformed the most widely used similarity scores in collaboritive filtering, a popular weighting method [98] (see Section 4.3.3) as well as the Battacharyya similarity score that was designed specifically for the extremely sparse setting. It would certainly be possible to combine LiRa with some of the aforementioned techniques in order to further improve recommendation accuracy and speed. However, we have demonstrated that LiRa is in itself an improvement over traditional and widely accepted scores used in recommender systems. This supports the assumption that a clustering structure underlies the user data of a recommender system, and warrants further exploration into the structural properties of clusters in recommender system data.

Together, my work on exploiting clustering structure in genetic map and recommender system data has been a step toward more efficient analysis of large-scale, discrete-valued data with copious amounts of missing values. I have described algorithms for clustering, data reduction, and similarity comparison in domains where data is abundant, but the profusion of missing entries prevent efficient analysis using standard data analysis tools. I have focused specifically on the underlying clustering structure of such data sets, which led to insights that enabled the design of fast algorithms for knowledge discovery. In the future, I would like to continue to focus on underlying clustering structure in large-scale data, and its influence on data analysis techniques.

The question of how underlying clustering structure in various application domains can inform new approaches to efficient data analysis opens up new avenues for future research. Given the multitude of clustering algorithms and definitions of what constitutes a good cluster [99] and the uncertainty as to whether a clustering result is meaningful [100], a natural future research direction would be to evaluate the *clusterability* of large-scale data, to determine whether a data set has underlying clustering structure at all. Clusterability has been defined in various ways [95, 101], but has not been thoroughly evaluated on real-world data. Evaluating the clusterability of real, large-scale data sets could establish a knowledge base of general, well-defined characteristics of clusters in real-world data. Drawing on this knowledge would benefit researchers and practitioners when deciding how to best model the underlying clustering structure of a data set.

In addition to the general notion of clusterability in large-scale data, one structural property that has to date been only marignally explored is the fractal dimension of a data set. The fractal dimension, which characterizes the extent to which a data set is self-similar, or self-repeating, has received some attention in the past due to its applicability to detecting various low-dimensional structures in high-dimensional data [25, 35, 36]. However, exploration of fractal dimension for data analysis has been limited, especially with regard to studying its usefulness for filling in or handling missing data. Like the fundamental resolution, a low fractal dimension can imply a large amount of redundancy in a data set [25]. Therefore, even with a large amount of missing data, it may be possible to design fast data analysis methods, such as clustering or data reduction, if it is likely that a data set has inherently low fractal dimension. It would be worthwhile to measure the fractal dimension of clusters in real-world, large-scale data sets.

In the context of recommender systems, the LiRa score (chapter 3) was shown to be useful for detecting user similarity in a k-nearest neighbor collaborative filtering approach to recommendation. Beyond evaluating similarity, a possible future research direction is

to incorporate LiRa into a fast clustering algorithm for clustering-based recommendation algorithms, which have been shown to improve the efficiency of recommendation without losing much accuracy [92, 102, 103].

The focus of thesis has been on leveraging underlying clustering structure for faster data analysis. I have described novel methods for knowledge discovery in the genetic mapping and recommender system application domains, with applicability to the broad class of large-scale, sparse, discrete-valued data sets. In future work, I plan to further study underlying clustering structure in large-scale data. My broad aim in this dissertation has been to further improve the analysis of large-scale data with many missing values, in order to advance state-of-the-art knowledge extraction methods on modern data sets.

Bibliography

[1] R. Xu and D. Wunsch, *Survey of clustering algorithms*, *IEEE Transactions on neural networks* **16** (2005), no. 3 645–678.

[2] A. K. Jain, M. N. Murty, and P. J. Flynn, *Data clustering: a review*, *ACM computing surveys (CSUR)* **31** (1999), no. 3 264–323.

[3] M. E. Newman, *Modularity and community structure in networks*, *Proceedings of the national academy of sciences* **103** (2006), no. 23 8577–8582.

[4] F. Lu, A. E. Lipka, J. Glaubitz, R. Elshire, J. H. Cherney, M. D. Casler, E. S. Buckler, and D. E. Costich, *Switchgrass genomic diversity, ploidy, and evolution: novel insights from a network-based snp discovery protocol*, *PLoS genetics* **9** (2013), no. 1 e1003215.

[5] J. Cheema and J. Dicks, *Computational approaches and software tools for genetic linkage map estimation in plants*, *Briefings in Bioinformatics* **10** (2009), no. 6 595–608.

[6] G. M. Downs, P. Willett, and W. Fisanick, *Similarity searching and clustering of chemical-structure databases using molecular property data*, *Journal of Chemical Information and Computer Sciences* **34** (1994), no. 5 1094–1102.

[7] A. Bender, H. Y. Mussa, R. C. Glen, and S. Reiling, *Molecular similarity searching using atom environments, information-based feature selection, and a naive bayesian classifier*, *Journal of chemical information and computer sciences* **44** (2004), no. 1 170–178.

[8] U. von Luxburg, *A tutorial on spectral clustering*, *Statistics and Computing* **17** (2007), no. 4 395–416.

[9] L.-Y. Hu, M.-W. Huang, S.-W. Ke, and C.-F. Tsai, *The distance function effect on k-nearest neighbor classification for medical datasets*, *SpringerPlus* **5** (2016), no. 1 1304.

[10] S. Oron, T. Dekel, T. Xue, W. T. Freeman, and S. Avidan, *Best-buddies similarity-robust template matching using mutual nearest neighbors*, *IEEE transactions on pattern analysis and machine intelligence* (2017).

[11] M. Faruqui, Y. Tsvetkov, P. Rastogi, and C. Dyer, *Problems with evaluation of word embeddings using word similarity tasks*, *arXiv preprint arXiv:1605.02276* (2016).

[12] B. K. Patra, R. Launonen, V. Ollikainen, and S. Nandi, *A new similarity measure using bhattacharyya coefficient for collaborative filtering in sparse data*, *Knowledge-Based Systems* **82** (2015) 163–177.

[13] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, *Item-based collaborative filtering recommendation algorithms*, in *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, ACM, 2001.

[14] K. Ø. Mikalsen, F. M. Bianchi, C. Soguero-Ruiz, and R. Jenssen, *Time series cluster kernel for learning similarities between multivariate time series with missing data*, *Pattern Recognition* **76** (2018) 569–581.

[15] J. T. Chi, E. C. Chi, and R. G. Baraniuk, *k-pod: A method for k-means clustering of missing data*, *The American Statistician* **70** (2016), no. 1 91–99.

[16] A. K. Jain, *Data clustering: 50 years beyond k-means*, *Pattern recognition letters* **31** (2010), no. 8 651–666.

[17] S. Lloyd, *Least squares quantization in pcm*, *IEEE transactions on information theory* **28** (1982), no. 2 129–137.

[18] L. Kaufman and P. Rousseeuw, *Clustering by means of medoids*. North-Holland, 1987.

[19] Z. Huang, *Extensions to the k-means algorithm for clustering large data sets with categorical values*, *Data Mining and Knowledge Discovery* **2** (Sep, 1998) 283–304.

[20] P. S. Bradley, O. L. Mangasarian, and W. N. Street, *Clustering via concave minimization*, in *Advances in neural information processing systems*, pp. 368–374, 1997.

[21] F. Lin and W. W. Cohen, *Power iteration clustering*, in *Proc. of ICML*, vol. 10, 2010.

[22] M. Belkin and P. Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, *Neural computation* **15** (2003), no. 6 1373–1396.

[23] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et. al.*, *A density-based algorithm for discovering clusters in large spatial databases with noise.*, in *Kdd*, vol. 96, pp. 226–231, 1996.

[24] A. Gionis, A. Hinneburg, S. Papadimitriou, and P. Tsaparas, *Dimension induced clustering*, in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 51–60, ACM, 2005.

[25] Y. W. Yu, N. M. Daniels, D. C. Danko, and B. Berger, *Entropy-scaling search of massive biological data*, *Cell systems* **1** (2015), no. 2 130–140.

[26] D. Arthur and S. Vassilvitskii, *k-means++: The advantages of careful seeding*, in *ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[27] R. J. Campello, D. Moulavi, and J. Sander, *Density-based clustering based on hierarchical density estimates*, in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 160–172, Springer, 2013.

[28] J. Gan and Y. Tao, *Dbscan revisited: mis-claim, un-fixability, and approximation*, in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 519–530, ACM, 2015.

[29] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, *Dbscan revisited, revisited: Why and how you should (still) use dbscan*, *ACM Transactions on Database Systems (TODS)* **42** (2017), no. 3 19.

[30] C. Ding and X. He, *K-means clustering via principal component analysis*, in *Proceedings of the twenty-first international conference on Machine learning*, p. 29, ACM, 2004.

[31] M. W. Mahoney and P. Drineas, *Cur matrix decompositions for improved data analysis*, *Proceedings of the National Academy of Sciences* **106** (2009), no. 3 697–702.

[32] B. Schölkopf, A. Smola, and K.-R. Müller, *Kernel principal component analysis*, in *International Conference on Artificial Neural Networks*, pp. 583–588, Springer, 1997.

[33] D. L. Donoho and C. Grimes, *Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data*, *Proceedings of the National Academy of Sciences* **100** (2003), no. 10 5591–5596.

[34] R. M. Gahar, O. Arfaoui, M. S. Hidri, and N. B.-H. Alouane, *Dimensionality reduction with missing values imputation*, *arXiv preprint arXiv:1707.00351* (2017).

[35] D. Barbará and P. Chen, *Using the fractal dimension to cluster datasets*, in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 260–264, ACM, 2000.

[36] M. Hoecker, K. L. Polsterer, S. D. Kügler, and V. Heuveline, *Clustering of complex data-sets using fractal similarity measures and uncertainties*, in *Computational Science and Engineering (CSE), 2015 IEEE 18th International Conference on*, pp. 82–91, IEEE, 2015.

[37] M. Metzker, *Sequencing technologies – the next generation*, *Nature Reviews Genetics* **11** (2009), no. 1 31–46.

[38] M. V. Rockman and L. Kruglyak, *Recombinational landscape and population genomics of caenorhabditis elegans*, *PLoS Genetics* **5** (2009), no. 3 e1000419.

[39] Y. Wu, P. Bhat, T. Close, and S. Lonardi, *Efficient and accurate construction of genetic linkage maps from the minimum spanning tree of a graph*, *PLoS Genet.* **4** (2008), no. 10.

[40] C. Desrosiers and G. Karypis, *A comprehensive survey of neighborhood-based recommendation methods*, in *Recommender systems handbook*, pp. 107–144. Springer, 2011.

[41] J. Herlocker, J. A. Konstan, and J. Riedl, *An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms*, *Information Retrieval* **5** no. 4 287–310.

[42] H. J. Ahn, *A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem*, *Information Sciences* **178** (2008), no. 1 37–51.

[43] V. Strnadova, A. Buluç, J. Chapman, J. R. Gilbert, J. Gonzalez, S. Jegelka, D. Rokhsar, and L. Oliker, *Efficient and accurate clustering for large-scale genetic mapping*, in *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, pp. 3–10, IEEE, 2014.

[44] V. Strnadová-Neeley, A. Buluç, J. Chapman, J. R. Gilbert, J. Gonzalez, and L. Oliker, *Efficient data reduction for large-scale genetic mapping*, in *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, pp. 126–135, ACM, 2015.

[45] P. Stam, *Construction of integrated genetic linkage maps by means of a new computer package: Join map*, *The Plant Journal* **3** (1993), no. 5 739–744.

[46] M. Bādoiu, S. Har-Peled, and P. Indyk, *Approximate clustering via core-sets*, in *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, (New York, NY, USA), pp. 250–257, ACM, 2002.

[47] D. Feldman and M. Langberg, *A unified framework for approximating and clustering data*, in *STOC*, 2011.

[48] U. von Luxburg, S. Bubeck, S. Jegelka, and M. Kaufmann, *Consistent minimization of clustering objective functions*, in *NIPS*, 2007.

[49] J. B. S. Haldane, *The combination of linkage values and the calculation of distances between the loci of linked factors*, *J Genet* **8** (1919), no. 29 299–309.

[50] D. Kosambi, *The estimation of map distances from recombination values*, *Annals of Eugenics* **12** (1943), no. 1 172–175.

[51] G. Margarido, A. Souza, and A. Garcia, *Onemap: software for genetic mapping in outcrossing species*, *Hereditas* **144** (2007), no. 3 78–79.

[52] P. Rastas, L. Paulin, I. Hanski, R. Lehtonen, and P. Auvinen, *Lep-MAP: fast and accurate linkage map construction for large SNP datasets*, *Bioinformatics* (2013) advance access.

[53] B. Jackson, P. Schnable, and S. Aluru, *Consensus genetic maps as median orders from inconsistent sources*, *IEEE Trans. on Comp. Biology and Bioinformatics* **5** (2008), no. 2.

[54] A. Kozik and R. Michelmore, *MadMapper and CheckMatrix – python scripts to infer orders of genetic markers and for visualization and validation of genetic maps and haplotypes*, in *Proceedings of the Plant and Animal Genome XIV Conference, San Diego*, 2006.

[55] E. S. Lander, P. Green, J. Abrahamson, A. Barlow, M. Daly, S. Lincoln, and L. Newberg, *MAPMAKER: an interactive computer package for constructing primary genetic linkage maps of experimental and natural populations*, *Genomics* **1** (1987) 174–181.

[56] A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams: Theory and practice*, *IEEE TKDE* **15** (2003), no. 3 515–528.

[57] M. Shindler, A. Wong, and A. Meyerson, *Fast and accurate k-means for large datasets*, in *NIPS*, 2011.

[58] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander, *OPTICS: ordering points to identify the clustering structure*, *ACM SIGMOD Record* **28** (1999), no. 2 49–60.

[59] M. Ester, H. Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, in *KDD*, vol. 96, 1996.

[60] T. Zhang, R. Ramakrishnan, and M. Livny, *Birch: An efficient data clustering method for very large databases*, 1996.

[61] S. Guha, R. Rastogi, and K. Shim, *CURE: an efficient clustering algorithm for large databases*, *ACM SIGMOD Record* **27** (1998), no. 2 73–84.

[62] N. Tinker, *Spaghetti: Simulation software to test genetic mapping programs*, *Journal of Heredity* **101** (2010), no. 2 257–259.

[63] S. Wagner and D. Wagner, *Comparing Clusterings – An Overview*. Universität Karlsruhe, Fakultät für Informatik, 2007.

[64] J. C. et al., *Chromosome-scale assembly of the hexaploid wheat genome from whole genome shotgun sequencing*, in *submission to Genome Biology*, 2014.

[65] J. Chapman, M. Mascher, A. Buluç, *et. al.*, *A whole-genome shotgun approach for assembling and anchoring the hexaploid bread wheat genome*, *Genome Biology* **16** (2015), no. 26.

[66] A. Auton, *The Estimation of Recombination Rates from Population Genetic Data*. PhD thesis, University of Oxford, 2007.

[67] M. Mascher, G. J. Muehlbauer, D. S. Rokhsar, J. Chapman, *et. al.*, *Anchoring and ordering NGS contig assemblies by population sequencing (POPSEQ)*, *The Plant Journal* **76** (2013), no. 4 718–727.

[68] V. Strnadova, A. Buluç, J. Chapman, J. Gilbert, J. Gonzalez, S. Jegelka, D. Rokshar, and L. Oliker, *Efficient and accurate clustering for large scale genetic mapping*, in *BIBM*, 2014.

[69] B. Collard and D. Mackill, *Marker-assisted selection: an approach for precision plant breeding in the twenty-first century*, *Philos Trans R Soc Lond B Biol Sci.* **363** (2008), no. 1491 557–572.

[70] Y. Li, C. Sidore, H. M. Kang, M. Boehnke, and G. R. Abecasis, *Low-coverage sequencing: implications for design of complex trait association studies*, *Genome research* (2011).

[71] E. J. Candès and B. Recht, *Exact matrix completion via convex optimization*, *Foundations of Computational mathematics* **9** (2009), no. 6 717–772.

[72] J. Xu, R. Wu, K. Zhu, B. Hajek, R. Srikant, and L. Ying, *Exact block-constant rating matrix recovery from a few noisy observations*, *arXiv preprint arXiv:1310.0512* (2013).

[73] S. Bailey, *Principal component analysis with noisy and/or missing data*, *Publications of the Astronomical Society of the Pacific* **124** (2012), no. 919 1015–1023.

[74] K. Q. Weinberger and L. K. Saul, *Unsupervised learning of image manifolds by semidefinite programming*, *International Journal of Computer Vision* **70** (2006), no. 1 77–90.

[75] E. Elhamifar and R. Vidal, *Sparse manifold clustering and embedding*, in *NIPS*, pp. 55–63, 2011.

[76] S. T. Roweis and L. K. Saul, *Nonlinear dimensionality reduction by locally linear embedding*, *Science* **290** (2000), no. 5500 2323–2326.

[77] D. Karakos, S. Khudanpur, J. Eisner, and C. E. Priebe, *Unsupervised classification via decision trees: An information-theoretic perspective*, in *ICASSP*, vol. 5, IEEE, 2005.

[78] F. Ricci, L. Rokach, and B. Shapira, *Introduction to recommender systems handbook*. Springer, 2011.

[79] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, *Evaluating collaborative filtering recommender systems*, *ACM Transactions on Information Systems (TOIS)* **22** (2004), no. 1 5–53.

[80] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, *Recommender systems survey*, *Knowledge-Based Systems* **46** (2013) 109–132.

[81] A. Gunawardana and G. Shani, *Evaluating recommender systems*, in *Recommender Systems Handbook*, pp. 265–308. Springer, 2015.

[82] Y. Koren and R. Bell, *Advances in collaborative filtering*, in *Recommender systems handbook*, pp. 77–118. Springer, 2015.

[83] T. Dunning, *Accurate methods for the statistics of surprise and coincidence*, *Computational linguistics* **19** (1993), no. 1 61–74.

[84] P. Casinelli, *Evaluating and implementing recommender systems as web services using apache mahout*, .

[85] D. Paraschakis, B. J. Nilsson, and J. Holländer, *Comparative evaluation of top-n recommenders in e-commerce: an industrial perspective*, .

[86] O. Jojic, M. Shukla, and N. Bhosarekar, *A probabilistic definition of item similarity*, in *Proceedings of the fifth ACM conference on Recommender systems*, pp. 229–236, ACM, 2011.

[87] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, *Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering*, in *Proceedings of the fifth international conference on computer and information technology*, vol. 1, Citeseer, 2002.

[88] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen, *Scalable collaborative filtering using cluster-based smoothing*, in *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 114–121, ACM, 2005.

[89] S. K. L. Al Mamunur Rashid, G. Karypis, and J. Riedl, *Clustknn: a highly scalable hybrid model-& memory-based cf algorithm*, *Proceeding of WebKDD* (2006).

[90] J. Das, P. Mukherjee, S. Majumder, and P. Gupta, *Clustering-based recommender system using principles of voting theory*, in *Contemporary Computing and Informatics (IC3I), 2014 International Conference on*, pp. 230–235, IEEE, 2014.

[91] J. Das, S. Majumder, D. Dutta, and P. Gupta, *Iterative use of weighted voronoi diagrams to improve scalability in recommender systems*, in *Advances in Knowledge Discovery and Data Mining*, pp. 605–617. Springer, 2015.

[92] S. Zahra, M. A. Ghazanfar, A. Khalid, M. A. Azam, U. Naeem, and A. Prugel-Bennett, *Novel centroid selection approaches for kmeans-clustering based recommender systems*, *Information Sciences* (2015).

[93] S. M. McNee, J. Riedl, and J. A. Konstan, *Being accurate is not enough: how accuracy metrics have hurt recommender systems*, in *CHI'06 extended abstracts on Human factors in computing systems*, pp. 1097–1101, ACM, 2006.

[94] G. Adomavicius and A. Tuzhilin, *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*, *IEEE transactions on knowledge and data engineering* **17** (2005), no. 6 734–749.

[95] M. Ackerman and S. Ben-David, *Clusterability: A theoretical study*, in *Artificial Intelligence and Statistics*, pp. 1–8, 2009.

[96] X. Amatriain, A. Jaimes, N. Oliver, and J. M. Pujol, *Data mining methods for recommender systems*, in *Recommender systems handbook*, pp. 39–71. Springer, 2011.

[97] D.-K. Chae, S.-C. Lee, S.-Y. Lee, and S.-W. Kim, *On identifying k-nearest neighbors in neighborhood models for efficient and effective collaborative filtering*, *Neurocomputing* **278** (2018) 134–143.

[98] R. M. Bell and Y. Koren, *Improved neighborhood-based collaborative filtering*, in *KDD cup and workshop at the 13th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 7–14, Citeseer, 2007.

[99] C. Hennig, *What are the true clusters?*, *Pattern Recognition Letters* **64** (2015) 53–62.

[100] C. Moore, *The computer science and physics of community detection: Landscapes, phase transitions, and hardness*, *arXiv preprint arXiv:1702.00467* (2017).

[101] S. Ben-David, *Computational feasibility of clustering under clusterability assumptions*, *arXiv preprint arXiv:1501.00437* (2015).

[102] T. George and S. Merugu, *A scalable collaborative filtering framework based on co-clustering*, in *Data Mining, Fifth IEEE international conference on*, pp. 4–pp, IEEE, 2005.

[103] G. Guo, J. Zhang, and N. Yorke-Smith, *Leveraging multiviews of trust and similarity to enhance clustering-based recommender systems*, *Knowledge-Based Systems* **74** (2015) 14–27.